TITLE:

Ternary Decision Diagrams and Their Applications for Logic Synthesis(Dissertation_全文)

AUTHOR(S):

Yasuoka, Koichi

CITATION:

Yasuoka, Koichi. Ternary Decision Diagrams and Their Applications for Logic Synthesis. 京都大学, 1997, 博士(工学)

ISSUE DATE:

1997-03-24

URL:

https://doi.org/10.11501/3123604

RIGHT:

京都大学学術情報リポジトリ
KURENAI 紅
Kyoto University Research Information Repository

新 制
工
1077

Ternary Decision Diagrams and Their Applications for Logic Synthesis

Koichi Yasuoka

November 1996

Ternary Decision Diagrams and Their Applications for Logic Synthesis

Koichi Yasuoka

November 1996

Abstract

Since Boolean algebra was introduced into the field of analysis of logic circuits, Boolean functions have occupied a fundamental position in the field of logic synthesis. In other words, the researches into methods to represent Boolean functions have become very important to design of logic circuits. With the recent advance in Very Large-Scale Integration technology, design of logic circuits grows out of manual control, and therefore, it is entrusted to automatic Computer-Aided Design systems. The performance of such CAD systems widely depends on how effective method to manipulate Boolean expressions they adopt, thus the researches on the representation of Boolean functions are the matters of weight in the field of VLSI design.

Sum-of-products forms are classical but fundamental idea to represent Boolean functions. Sum-of-products forms are regarded as a special case of Boolean expressions that are actualized in a two-level AND-OR structure such as Programmable Logic Array. The optimization problem of sum-of-products forms is applicable to the optimization of the sizes of PLAs, and various CAD systems to optimize sum-of-products forms have been developed. However, these conventional CAD systems manipulate sum-of-products forms with linked-list structure, thus several operations between sum-of-products forms consume enormous time and space in these systems. An efficient method to manipulate sum-of-products forms is desired.

Ringsum-of-products forms were originally from the mathematical representation of Boolean functions, and now they are regarded as a special case of Boolean expressions that

are actualized in a two-level AND-XOR structure. The researches on the minimization of ringsum-of-products forms have been developed from a rather theoretical point of view than that of sum-of-products forms, and several CAD systems to optimize ringsum-of-products forms have been developed. However, these CAD systems manipulate ringsum-of-products forms with Binary Decision Diagrams or their extensions, thus these systems can manipulate only 2^{2^n} of 2^{3^n} ringsum-of-products forms. It is too powerless, and an efficient method to represent ringsum-of-products forms is desired.

This thesis proposes Ternary Decision Diagrams as a graph representation method for both sum-of-products forms and ringsum-of-products forms. This thesis discusses theoretical aspects of Ternary Decision Diagrams, their implementations, and their applications on logic synthesis. Chapters 2 to 4 discuss about Ternary Decision Diagrams themselves, chapter 5 discusses the implementation of the Ternary Decision Diagrams Library, and chapter 6 discusses the applications.

In chapter 2, we define Ternary Decision Diagrams from the mathematical point of view. Then we discuss how the sets of products are represented on Ternary Decision Diagrams. Brief definitions of Binary Decision Diagrams and Quasi-reduced Binary Decision Diagrams are mentioned there.

In chapters 3 and 4, we discuss about the set operations on TDDs, which are essential operations on sum-of-products forms and ringsum-of-products forms, respectively. Then, we discuss about combinatorial circuits manipulations on Ternary Decision Diagrams using the set operations for sum-of-products forms and ringsum-of-products forms. We also discuss about the conversion between Ternary Decision Diagrams and Binary Decision Diagrams in the chapter.

In chapter 5, we discuss about the implementation techniques of the Ternary Decision Diagrams Library. Experimental results to compare the sizes between Ternary Decision Diagrams and Binary Decision Diagrams are also shown in the chapter.

In chapter 6, we apply Ternary Decision Diagrams for three problems in the field of logic synthesis. First, we discuss about an effective method to generate prime implicants

on sum-of-products Ternary Decision Diagrams. Second, we discuss about an effective method to generate neighfunctions on sum-of-products Ternary Decision Diagrams. Third, we discuss about an effective method to generate an optimized ringsum-of-products forms with Ternary Decision Diagrams using the set operations for ringsum-of-products forms.

Throughout this thesis, we realize that Ternary Decision Diagrams are very effective method to manipulate Boolean functions, and that the research on Ternary Decision Diagrams will contribute the development of the researches into logic synthesis.

Contents

Chapter 1

Introduction

1.1 Background

Since Boolean algebra was introduced into the field of analysis of logic circuits by Nakajima, Hanzawa, and Shannon [1, 2], Boolean functions have occupied a fundamental position in the field of logic synthesis. In other words, the researches into effective methods to represent Boolean functions have become very important to design of logic circuits. With the recent advance in Very Large-Scale Integration technology, the design of logic circuits grows out of manual control, and therefore, it is entrusted to automatic Computer-Aided Design systems. The performance of such CAD systems widely depends on how effective method to manipulate Boolean functions they adopt, thus the researches on the representation of Boolean functions are the matters of weight in the field of VLSI design.

Sum-of-products forms (also called cube sets, PLA forms, or two-level AND-OR logics) are classical but fundamental idea to represent Boolean functions introduced by Quine [3]. Sum-of-products forms are regarded as a special case of Boolean expressions that are realized in a two-level AND-OR structure such as Programmable Logic Array [9]. The optimization problem of sum-of-products forms [6, 7] is applicable to the optimization of the sizes of PLAs, and various CAD systems to optimize sum-of-products forms have

been developed [10, 13, 14, 15]. However, these conventional CAD systems manipulate sum-of-products forms with linked-list structure, thus several operations between sum-of-products forms consume enormous time and space in these systems. An efficient method to manipulate sum-of-products forms is desired.

Ringsum-of-products forms (also called Reed-Muller expansions, exclusive-or-sum-of-products forms, or two-level AND-XOR logics) were originally from the mathematical representation of Boolean functions by Reed and Muller [4, 5], and now ringsum-of-products forms are regarded as a special case of Boolean expressions that are realized in a two-level AND-XOR structure [8]. The researches on the minimization of ringsum-of-products forms has been developed from a rather theoretical point of view [12, 17] than that of sum-of-products forms, and several CAD systems to optimize ringsum-of-products forms have been developed [21, 26, 29]. However, these CAD systems manipulate ringsum-of-products forms with Binary Decision Diagrams (mentioned later) or their extensions, thus these systems can manipulate only 2^{2^n} of 2^{3^n} ringsum-of-products forms. It is too powerless, and an efficient method to manipulate ringsum-of-products forms is desired.

Binary Decision Diagrams were invented by Akers [11] to represent Boolean functions. Then a very efficient method to manipulate Boolean functions on BDDs was developed by Bryant [16]. Thereafter, BDDs have become very popular in the field of VLSI logic system designs, and have occupied very high position at implementing CAD systems [18, 19, 26, 31]. However, BDDs represent Boolean functions, and cannot represent sum-of-products forms or ringsum-of-products forms without some extensions [24, 30]. In other words, BDDs are rather powerless to represent sets with binate literals, and new idea to represent sets of products is required.

Ternary Decision Diagrams were first introduced to represent Pseudo Kronecker Expressions [23, 25], which are subclass of ringsum-of-products forms, and were then used to manipulate some subclass of sum-of-products forms [28]. Nowadays a very efficient method to manipulate both sum-of-products forms and ringsum-of-products forms on TDDs has been proposed by the author [33]. With the method, Ternary Decision Diagrams acquire

properties:

- to represent sets of products uniquely,
- to manipulate sum-of-products forms efficiently, and
- to manipulate ringsum-of-products forms efficiently.

Furthermore, the operations such as Cartesian products or weak division of sets of products, which are hardly implementable on the conventional CAD systems, are very easily realized on Ternary Decision Diagrams. The research on Ternary Decision Diagrams will contribute the development of CAD systems.

1.2 Overview of Thesis

This thesis discusses theoretical aspects of Ternary Decision Diagrams, their implementations, and their applications on logic synthesis. Chapters 2 to 4 discuss about Ternary Decision Diagrams themselves, chapter 5 discusses the implementation of the Ternary Decision Diagrams Library, and chapter 6 discusses the applications.

In chapter 2, we define Ternary Decision Diagrams from the mathematical point of view. Then, we discuss how the sets of products are represented on Ternary Decision Diagrams. Both sum-of-products forms and ringsum-of-products forms can be considered as the sets of products, thus in the chapter we discuss that Ternary Decision Diagrams uniquely represent these forms. Also in the chapter we mention about the brief definitions of Binary Decision Diagrams and Quasi-reduced Binary Decision Diagrams.

In chapter 3, we first discuss about the set operations on TDDs, which are essential operations on sum-of-products forms, including union, Cartesian product, weak division, intersection, difference, and complement. All of these operations are defined recursively, and they are very easily implementable. Then, we discuss about combinatorial circuits manipulations on Ternary Decision Diagrams using the set operations for sum-of-products

forms. Namely, in the chapter we discuss how we manipulate AND-, OR-, XOR-, and NOT-gates in the circuits under the sum-of-products Ternary Decision Diagrams. We also discuss about the conversion between sum-of-products Ternary Decision Diagrams and Binary Decision Diagrams.

In chapter 4, we first discuss about the set operations on TDDs, which are essential operations on ringsum-of-products forms, including symmetric difference, ringsum product, ringsum weak division, and ringsum complement. All of these operations are defined recursively, and they are very easily implementable. Then, we discuss about combinatorial circuits manipulations on Ternary Decision Diagrams using the set operations for ringsum-of-products forms. In other words, we discuss how we manipulate AND-, XOR-, OR-, and NOT-gates in the circuits under the ringsum-of-products Ternary Decision Diagrams. We also discuss about the conversion between ringsum-of-products Ternary Decision Diagrams and Binary Decision Diagrams.

In chapter 5, we discuss about the implementation techniques of the Ternary Decision Diagrams Library. The real structure of the nodes of Ternary Decision Diagrams is mentioned, and the effectiveness of the operation-result table is discussed here. Also, we consider some skillful mechanisms to guarantee canonicals of Ternary Decision Diagrams. Experimental results to compare the sizes between Ternary Decision Diagrams and Binary Decision Diagrams are shown in the chapter.

In chapter 6, we apply Ternary Decision Diagrams for three problems in the field of logic synthesis. First, we discuss about an effective method to generate prime implicants [7] on sum-of-products Ternary Decision Diagrams. The generation of the prime implicants is a fundamental problem in the field of logic design, and is required at so many scenes in logic synthesis. Second, we discuss about an effective method to generate neighfunctions [20] on sum-of-products Ternary Decision Diagrams. The neighfunction is the sum of implicants which is subsumed by a particular minterm, and is used in a PLA-optimizer "TACCO" [22]. Third, we discuss about an effective method to generate an optimized ringsum-of-products forms with Ternary Decision Diagrams using the set

operations for ringsum-of-products forms. Among the set operations, weak division is the most significant operation for the optimization.

Chapter 7 concludes this thesis.

Chapter 2

Preliminaries

2.1 Ternary Decision Diagrams

Ternary Decision Diagrams are direct 3-degree acyclic graphs denoted by an 8-tuple $\langle V, \boxed{0}, \boxed{1}, \epsilon_0, \epsilon_1, \epsilon_*, X, \lambda\rangle$; where $\boxed{0}$ and $\boxed{1}$ are leaf nodes, V is a set of non-leaf nodes, ϵ_0, ϵ_1, and ϵ_* are the edges, i.e. mappings from V to $V \cup \{\boxed{0}, \boxed{1}\}$, $X \equiv \{x_1, x_2, \cdots, x_n\}$ is a set of variables, and λ is the labeling of the non-leaf nodes that is a mapping from V to X; and TDDs must satisfy following three rules:

i) **Variable Ordering Rule** (see Figure 2.1)

There exists a total order $x_{\theta_1} > x_{\theta_2} > \cdots > x_{\theta_n}$ on X such that

$$\forall v \in V, \begin{cases} \epsilon_0(v) \in V \Rightarrow \lambda(v) > \lambda(\epsilon_0(v)), \\ \epsilon_1(v) \in V \Rightarrow \lambda(v) > \lambda(\epsilon_1(v)), \text{and} \\ \epsilon_*(v) \in V \Rightarrow \lambda(v) > \lambda(\epsilon_*(v)). \end{cases}$$

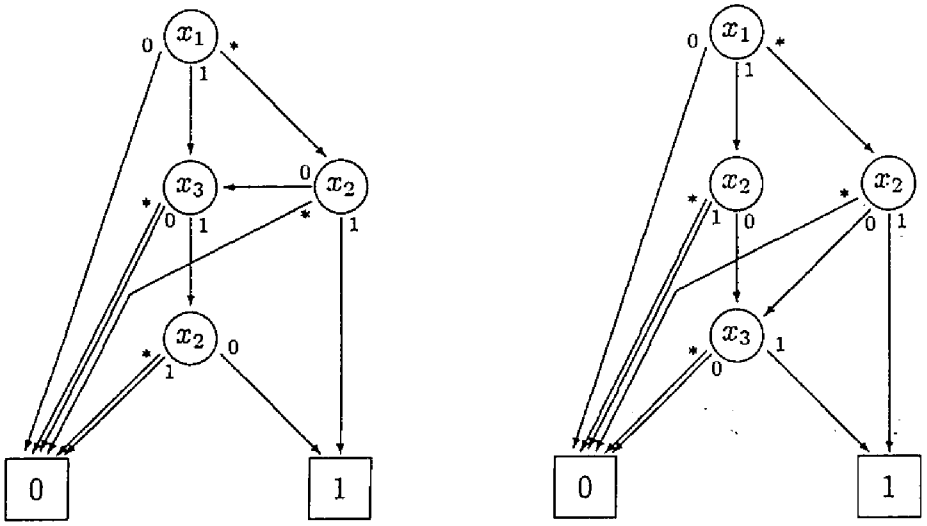
ii) **Node Reduction Rule** (see Figure 2.2)

$\forall v \in V,\ \epsilon_0(v) \neq \boxed{0} \vee \epsilon_1(v) \neq \boxed{0}$.

iii) **Node Unification Rule** (see Figure 2.3)

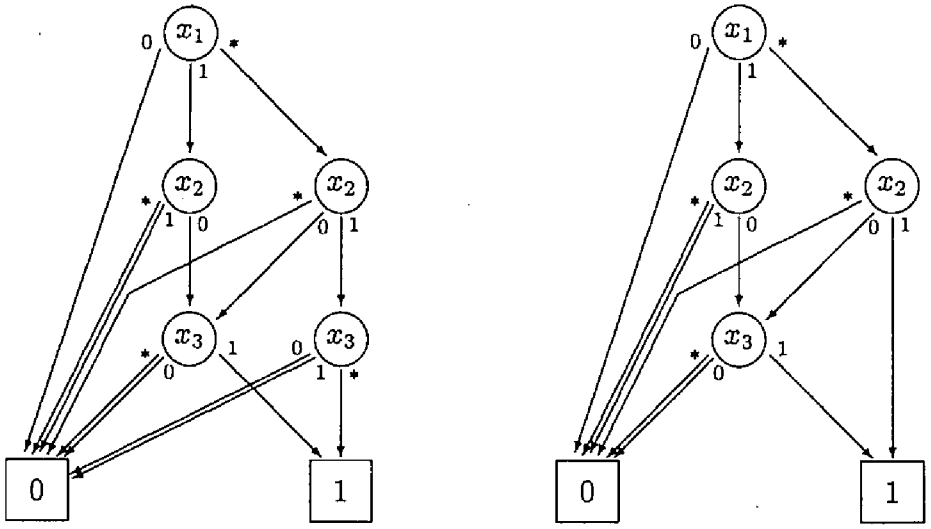
$\forall v, v' \in V,\ \lambda(v) = \lambda(v') \wedge \epsilon_0(v) = \epsilon_0(v') \wedge \epsilon_1(v) = \epsilon_1(v') \wedge \epsilon_*(v) = \epsilon_*(v') \Rightarrow v = v'$.

We define a set of products represented by a node in TDDs as follows:



(a) illegal TDD (b) legal TDD

Figure 2.1: Variable Ordering Rule on TDDs



(a) illegal TDD (b) legal TDD

Figure 2.2: Node Reduction Rule on TDDs

- $\boxed{0}$ represents the empty set $\emptyset$.
- $\boxed{1}$ represents the set $\{1\}$.
- A non-leaf node $v \in V$ represents the union of the following three sets; the set of the logical product of $\overline{\lambda(v)}$ and every product in the set represented by $\epsilon_0(v)$, the set of the logical product of $\lambda(v)$ and every product in the set represented by $\epsilon_1(v)$, and the set represented by $\epsilon_*(v)$.

It is sure on this definition that any two different nodes in TDDs represent two different sets of products, and that any set of products can be represented by its corresponding node in TDDs. In other words, nodes of TDDs are canonical for sets of products. Figure 2.4 shows an example of sets of products represented by nodes in TDDs. In the rest of this thesis, we think of $\emptyset$, $\{1\}$, $\{\overline{x_i}\}$, and $\{x_i\}$ $(x_i \in X)$ as basic sets that are given in TDDs from the beginning.

In the next chapter, we regard that TDDs represent sum-of-products forms by sets of products. In chapter 4, separately from chapter 3, we regard that TDDs represent ringsum-of-products forms by sets of products.

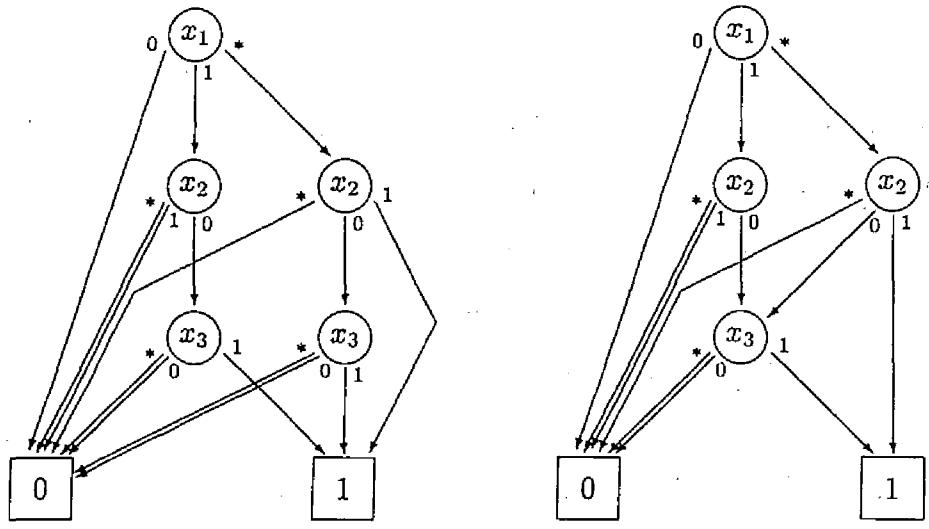
2.2 Binary Decision Diagrams

Binary Decision Diagrams [11, 16] are direct 2-degree acyclic graphs denoted by a 7-tuple $\langle V, \boxed{0}, \boxed{1}, \epsilon_0, \epsilon_1, X, \lambda \rangle$; where $\boxed{0}$ and $\boxed{1}$ are leaf nodes, V is a set of non-leaf nodes, ϵ_0 and ϵ_1 are the edges, i.e. mappings from V to $V \cup \{\boxed{0}, \boxed{1}\}$, $X \equiv \{x_1, x_2, \cdots, x_n\}$ is a set of variables, and λ is the labeling of the non-leaf nodes that is a mapping from V to X; and BDDs must satisfy following three rules:

i) **Variable Ordering Rule**

There exists a total order $x_{\theta_1} > x_{\theta_2} > \cdots > x_{\theta_n}$ on X such that

$$\forall_{v \in V}, \begin{cases} \epsilon_0(v) \in V \Rightarrow \lambda(v) > \lambda(\epsilon_0(v)) \text{ and} \\ \epsilon_1(v) \in V \Rightarrow \lambda(v) > \lambda(\epsilon_1(v)). \end{cases}$$



(a) illegal TDD (b) legal TDD

Figure 2.3: Node Unification Rule on TDDs

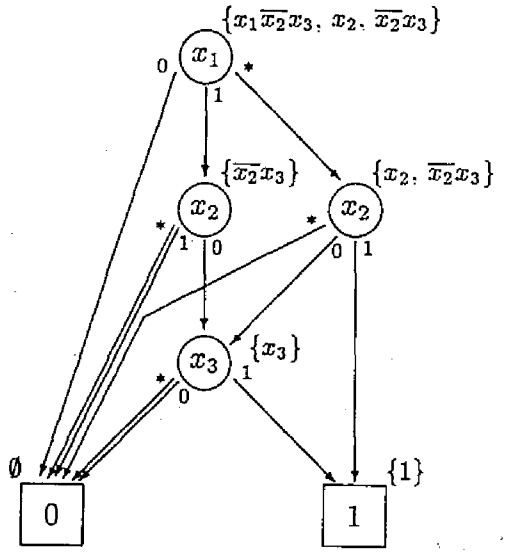


Figure 2.4: Sets of products represented in TDDs

ii) **Node Reduction Rule**

$^{\forall}v \in V,\ \epsilon_0(v) \neq \epsilon_1(v)$.

iii) **Node Unification Rule**

$^{\forall}v,v' \in V,\ \lambda(v)=\lambda(v') \wedge \epsilon_0(v)=\epsilon_0(v') \wedge \epsilon_1(v)=\epsilon_1(v') \Rightarrow v=v'$.

We define a Boolean function represented by a node in BDDs as follows:

- $\boxed{0}$ represents the Boolean function 0 (so-called inconsistency).
- $\boxed{1}$ represents the Boolean function 1 (so-called tautology).
- A non-leaf node $v \in V$ represents the logical sum of the following two Boolean functions; the logical product of $\overline{\lambda(v)}$ and the Boolean function represented by $\epsilon_0(v)$, and the logical product of $\lambda(v)$ and the Boolean function represented by $\epsilon_1(v)$.

It is sure on this definition that any two different nodes in BDDs represent two different Boolean functions, and that any Boolean function can be represented by its corresponding node in BDDs. In other words, nodes of BDDs are canonical for Boolean functions.

2.3 Quasi-reduced Binary Decision Diagrams

Quasi-reduced Binary Decision Diagrams [32] are direct 2-degree acyclic graphs denoted by a 7-tuple $\langle V, \boxed{0}, \boxed{1}, \epsilon_0, \epsilon_1, X, \lambda \rangle$; where $\boxed{0}$ and $\boxed{1}$ are leaf nodes, V is a set of non-leaf nodes, ϵ_0 and ϵ_1 are the edges, i.e. mappings from V to $V \cup \{\boxed{0}, \boxed{1}\}$, $X \equiv \{x_1, x_2, \cdots, x_n\}$ is a set of variables, and λ is the labeling of the non-leaf nodes that is a mapping from V to X; and QBDDs must satisfy following two rules:

i) **Strong Variable Ordering Rule**

There exists a total order $x_{\theta_1} > x_{\theta_2} > \cdots > x_{\theta_n}$ on X such that

$$^{\forall}v \in V,\ \begin{cases} \lambda(v)=x_{\theta_i}\ (i \neq n) \Rightarrow \lambda(\epsilon_0(v))=\lambda(\epsilon_1(v))=x_{\theta_{i+1}} \text{ and} \\ \lambda(v)=x_{\theta_n} \Rightarrow \epsilon_0(v), \epsilon_1(v) \in \{\boxed{0}, \boxed{1}\} \end{cases}$$

ii) **Node Unification Rule**

$^{\forall}v,v' \in V,\ \lambda(v)=\lambda(v') \wedge \epsilon_0(v)=\epsilon_0(v') \wedge \epsilon_1(v)=\epsilon_1(v') \Rightarrow v=v'$.

We define a Boolean function represented by a node in QBDDs as follows:

- $\boxed{0}$ represents the Boolean function 0 (so-called inconsistency).
- $\boxed{1}$ represents the Boolean function 1 (so-called tautology).
- A non-leaf node $v \in V$ represents the logical sum of the following two Boolean functions; the logical product of $\overline{\lambda(v)}$ and the Boolean function represented by $\epsilon_0(v)$, and the logical product of $\lambda(v)$ and the Boolean function represented by $\epsilon_1(v)$.

It is sure on this definition that any two different nodes in QBDDs represent two different Boolean functions, and that any Boolean function can be represented by its corresponding node in QBDDs. In other words, nodes of QBDDs are canonical for Boolean functions.

Chapter 3

Sum-of-Products Operations on TDDs

3.1 Introduction

When a set of products is given, it can be regarded either as a sum-of-products form or as a ringsum-of-products form. For example, the set $\{x_1\overline{x_2}x_3,\ x_2,\ \overline{x_2}x_3\}$ can be regarded either as $x_1\overline{x_2}x_3+x_2+\overline{x_2}x_3$ or as $x_1\overline{x_2}x_3\oplus x_2\oplus\overline{x_2}x_3$. Hence, when we represent a set of products in TDDs, we carry out different operations on TDDs according as our view of the set, namely, the sum-of-products form or the ringsum-of-products form.

In this chapter, we regard that Ternary Decision Diagrams represent sum-of-products forms by sets of products. In other words, all TDDs appearing in this chapter are for sum-of-products forms, and all set operations are defined as they are used on operations on sum-of-products forms. Figure 3.1 shows an example of sum-of-products forms represented by TDD.

We conjecture all procedures mentioned in this chapter have worst case time complexity $O(|R|)$ where $|R|$ is the worst size of resulting TDDs, assuming that the operation at each node requires constant time and that the same operation is never computed twice

using the operation-result table [16].

3.2 Sum-of-Products Operations

Here we consider essential operations on TDDs in which we represent sum-of-products forms. In other words, we think about "addition," "multiplication," and "logical not" in the field of sum-of-products forms.

3.2.1 Addition of SOP Forms

In this section, we consider "addition" of two sum-of-products forms. Namely we consider how to realize, for example, "$x_2+\overline{x_1}x_3$ added to $x_1\overline{x_2}x_3+x_2+\overline{x_2}x_3$ gives $x_1\overline{x_2}x_3+x_2+\overline{x_2}x_3+\overline{x_1}x_3$" on Ternary Decision Diagrams.

The set operator **union**, denoted by $\cup$, can actualize the addition of two sum-of-products forms. For example, $\{x_1\overline{x_2}x_3,\ x_2,\ \overline{x_2}x_3\}\cup\{x_2,\ \overline{x_1}x_3\}=\{x_1\overline{x_2}x_3,\ x_2,\ \overline{x_2}x_3,\ \overline{x_1}x_3\}$ can be regarded as $(x_1\overline{x_2}x_3+x_2+\overline{x_2}x_3)+(x_2+\overline{x_1}x_3)=x_1\overline{x_2}x_3+x_2+\overline{x_2}x_3+\overline{x_1}x_3$. We can realize the procedure for the operator $\cup$ on TDDs as shown in Figure 3.2. The procedure is mainly based on the following formula (where the forms A to F include neither $\overline{x_i}$ nor x_i, and $\overline{x_i}$A denotes the set consisting of the products of $\overline{x_i}$ and all elements in A):

$$(\overline{x_i}\mathrm{A}\cup x_i\mathrm{B}\cup\mathrm{C})\cup(\overline{x_i}\mathrm{D}\cup x_i\mathrm{E}\cup\mathrm{F})=\overline{x_i}(\mathrm{A}\cup\mathrm{D})\cup x_i(\mathrm{B}\cup\mathrm{E})\cup(\mathrm{C}\cup\mathrm{F})$$

and it terminates when it reaches to the leaf nodes, where the rules $\mathrm{P}\cup\emptyset=\mathrm{P}$, $\emptyset\cup\mathrm{Q}=\mathrm{Q}$, and $\{1\}\cup\{1\}=\{1\}$ are used.

3.2.2 Multiplication of SOP Forms

Here we consider "multiplication" of two sum-of-products forms. Namely we consider how to realize, for example, "$x_1\overline{x_2}x_3+x_2+\overline{x_2}x_3$ multiplied by $x_1+\overline{x_3}$ gives $x_1\overline{x_2}x_3+x_1x_2+x_2\overline{x_3}$" on Ternary Decision Diagrams.

The set operator **Cartesian product**, denoted by $\times$, can actualize the multiplication of two sum-of-products forms. For example, $\{x_1\overline{x_2}x_3,\ x_2,\ \overline{x_2}x_3\}\times\{x_1,\ \overline{x_3}\}=\{x_1\overline{x_2}x_3,\ x_1x_2,\ x_2\overline{x_3}\}$ can be regarded as $(x_1\overline{x_2}x_3+x_2+\overline{x_2}x_3)(x_1+\overline{x_3})=x_1\overline{x_2}x_3+x_1x_2+x_2\overline{x_3}$. We can realize the procedure for the operator $\times$ on TDDs as shown in Figure 3.3. The procedure is based on the following formula:

$$(\overline{x_i}\mathrm{A}\cup x_i\mathrm{B}\cup\mathrm{C})\times(\overline{x_i}\mathrm{D}\cup x_i\mathrm{E}\cup\mathrm{F})=$$
$$\overline{x_i}((\mathrm{A}\times\mathrm{D})\cup(\mathrm{A}\times\mathrm{F})\cup(\mathrm{C}\times\mathrm{D}))\cup x_i((\mathrm{B}\times\mathrm{E})\cup(\mathrm{B}\times\mathrm{F})\cup(\mathrm{C}\times\mathrm{E}))\cup(\mathrm{C}\times\mathrm{F})$$

and it terminates when it reaches to the leaf nodes, where the rules $\mathrm{P}\times\emptyset=\emptyset$ and $\mathrm{P}\times\{1\}=\mathrm{P}$ are used.

Then we consider an inverse operation for the multiplication of sum-of-products forms. In other words, we consider "division" on the field of sum-of-products forms.

In order to make the "division" one-valued, we define a set operation named **weak division**, denoted by /, as described below:

> When sets of products P and Q are given, the quotient $\mathrm{R}=\mathrm{P}/\mathrm{Q}$ is the maximum set that satisfies $\mathrm{R}\times\mathrm{Q}\subseteq\mathrm{P}$, where R does not include variables appeared in Q.

For example, $\{x_1\overline{x_2}x_3,\ x_2,\ \overline{x_2}x_3\}/\{1,\ x_1\}=\{\overline{x_2}x_3\}$. We can realize the procedure for the operator / on TDDs as shown in Figure 3.4. The procedure is based on the following formula ($\cap$ denotes the intersection of sets of products):

$$(\overline{x_i}\mathrm{A}\cup x_i\mathrm{B}\cup\mathrm{C})/(\overline{x_i}\mathrm{D}\cup x_i\mathrm{E}\cup\mathrm{F})=(\mathrm{A}/\mathrm{D})\cap(\mathrm{B}/\mathrm{E})\cap(\mathrm{C}/\mathrm{F})$$

when $\mathrm{D}\neq\emptyset$ or $\mathrm{E}\neq\emptyset$. Otherwise we use the formula

$$(\overline{x_i}\mathrm{A}\cup x_i\mathrm{B}\cup\mathrm{C})/\mathrm{F}=\overline{x_i}(\mathrm{A}/\mathrm{F})\cup x_i(\mathrm{B}/\mathrm{F})\cup(\mathrm{C}/\mathrm{F})\ .$$

The procedure terminates when the rule $\mathrm{P}/\emptyset=\infty$ or $\mathrm{P}/\{1\}=\mathrm{P}$ is used, where ∞ means the universal set which satisfies $\infty\cap\mathrm{R}=\mathrm{R}$ for any set R.

In order to implement the weak division procedure, we need the set operation **intersection**, denoted by $\cap$, on TDDs. The procedure for the operator $\cap$ on TDDs (shown in

Figure 3.5) is similar to the one for $\cup$ save that its termination rules are $P \cap \emptyset = \emptyset$, $\emptyset \cap Q = \emptyset$, and $\{1\} \cap \{1\} = \{1\}$.

Lastly, we consider getting a remainder after a weak division. In other words, after $x_1\overline{x_2}x_3 + x_2 + \overline{x_2}x_3$ divided by $1 + x_1$ gives the quotient $\overline{x_2}x_3$, we should think how to get the remainder x_2.

The set operator **difference**, denoted by $-$, is suitable for the purpose, since we can get the remainder with $P - ((P/Q) \times Q)$ as a set of products. The procedure for the operator $-$ on TDDs (shown in Figure 3.6) is similar to the one for $\cup$ save that its termination rules are $P - \emptyset = P$, $\emptyset - Q = \emptyset$, and $\{1\} - \{1\} = \emptyset$.

3.2.3 Logical Not of SOP Forms

When a sum-of-products form F is given, we often need a "logical not" sum-of-products form F′ of F which satisfies:

- F added to F′ gives a sum-of-products form of the Boolean function 1, and
- F multiplied by F′ gives the sum-of-products form of the Boolean function 0.

Here we define an unary set operation **complement**, denoted by $\widetilde{\ }$, to get one of such "logical not" sum-of-products form as a set of products. Figure 3.7 shows the procedure for the operator $\widetilde{\ }$ on TDDs. The procedure is based on the following formula:

$$(\overline{x_i}A \cup \widetilde{x_i}B \cup C) = (x_i \cup \tilde{A})(\overline{x_i} \cup \tilde{B})\tilde{C} = \overline{x_i}(\tilde{A}\tilde{C}) \cup x_i(\tilde{B}\tilde{C}) \cup (\tilde{A}\tilde{B}\tilde{C}) =$$
$$\overline{x_i}(\widetilde{A \cup C}) \cup x_i(\widetilde{B \cup C}) \cup (\widetilde{A \cup B \cup C})$$

and it terminates when it reaches to the leaf nodes, where the rules $\tilde{\emptyset} = \{1\}$ and $\widetilde{\{1\}} = \emptyset$ are used.

3.3 Combinatorial Circuits Manipulation Using SOP-TDDs

In this section, we mention about the method to generate representation of outputs of all gates in combinatorial circuits under the sum-of-products Ternary Decision Diagrams. Here we assume that the circuits consist of 2-input AND-gates, 2-input OR-gates, 2-input XOR-gates, and 1-input NOT-gates. We also assume that primary inputs are represented by given sets of $\{x_i\}$.

i) **2-input AND-gate**

When a 2-input AND-gate, whose inputs are represented in sum-of-products forms P and Q, is given, we generate P×Q for its output as shown in Figure 3.8. For example, when the two inputs of an AND-gate are represented in $x_1\overline{x_2}x_3+x_2+\overline{x_2}x_3$ and $x_1+\overline{x_3}$, its output results in $x_1\overline{x_2}x_3+x_1x_2+x_2\overline{x_3}$. TDDs actualize this operation as the Cartesian product procedure mentioned in the previous section.

ii) **2-input OR-gate**

When a 2-input OR-gate, whose inputs are represented in sum-of-products forms P and Q, is given, we generate P∪Q for its output as shown in Figure 3.9. For example, when the two inputs of an OR-gate are represented in $x_1\overline{x_2}x_3+x_2+\overline{x_2}x_3$ and $x_2+\overline{x_1}x_3$, its output results in $x_1\overline{x_2}x_3+x_2+\overline{x_2}x_3+\overline{x_1}x_3$. TDDs actualize this operation as the union procedure mentioned in the previous section.

iii) **2-input XOR-gate**

When a 2-input XOR-gate, whose inputs are represented in sum-of-products forms P and Q, is given, we generate $(P\cup\tilde{Q})\times(\tilde{P}\cup Q)$ for one of the sum-of-products forms of its output as shown in Figure 3.10. For example, when the two inputs of an XOR-gate are represented in $x_1\overline{x_2}x_3+x_2+\overline{x_2}x_3$ and $x_2+\overline{x_1}x_3$, its output results in $x_2+\overline{x_1}x_2x_3+\overline{x_1}\,\overline{x_2}x_3+\overline{x_1}\,\overline{x_2}\,\overline{x_3}+x_1\overline{x_2}\,\overline{x_3}$. TDDs actualize this operation as the combi-

nation of the complement procedure, the union procedure, and the Cartesian product procedure mentioned in the previous section.

iv) NOT-gate

When a NOT-gate, whose input is represented in sum-of-products form P, is given, we generate $\tilde{P}$ for its output as shown in Figure 3.11. For example, when the input of a NOT-gate is represented in $x_2+\overline{x_1}x_3$, its output results in $x_1\overline{x_2}+\overline{x_1}\,\overline{x_2}\,\overline{x_3}+\overline{x_2}\,\overline{x_3}$. TDDs actualize this operation as the complement procedure mentioned in the previous section.

3.4 Conversion between SOP-TDDs and BDDs

In this section, we consider a method to convert a Binary Decision Diagram into a sum-of-products Ternary Decision Diagram which represents the same Boolean function represented by the BDD, vice versa.

When a Binary Decision Diagram is given, we can obtain a sum-of-products Ternary Decision Diagram which represents the same Boolean function represented by the given BDD, adding $\boxed{0}$-directed $*$-edges to all non-leaf nodes of the BDD. For example, the Binary Decision Diagram in Figure 3.12(a) is given, we can obtain the Ternary Decision Diagram in (b). They both represent the same Boolean function $\overline{x_1}x_2\overline{x_3}+\overline{x_1}\,\overline{x_2}+x_1\overline{x_2}\,\overline{x_3}+x_1x_2$. This is because all the products, which we can derive from all paths from the top node to $\boxed{1}$ in a Binary Decision Diagram, are disjointed with one another. Therefore, the Boolean function represented by the BDD is regarded as the sum of the products, and it is easily represented by the SOP-TDD whose 0-edges and 1-edges constitutes the same graph of the BDD and whose $*$-edges direct $\boxed{0}$.

Inversely, when a sum-of-products Ternary Decision Diagram is given, we can obtain a Binary Decision Diagram which represents the same Boolean function represented by the given SOP-TDD, as follows:

Step.1 Generate Ternary Decision Diagram which represents the set of minterms, where the sum of the minterms expresses the same Boolean function as the given SOP-TDD. In order to generate such "sum-of-minterms" TDD Q from the given SOP-TDD P, we use Shannon Expansion as described below:

$$Q = \{\overline{x_1}, x_1\}\times\{\overline{x_2}, x_2\}\times\cdots\times\{\overline{x_n}, x_n\}\times P \ .$$

Step.2 Remove nodes whose 0-edge and 1-edge point the same node. In order to remove x_i-labeled ones of such "redundant" nodes from the Ternary Decision Diagram Q and obtain the new Ternary Decision Diagram Q', we use the following formula:

$$Q' = Q\cup(Q/\{\overline{x_i}, x_i\})-(Q/\{\overline{x_i}, x_i\})\times\{\overline{x_i}, x_i\} \ .$$

We can remove all nodes whose 0-edge and 1-edge point the same node by applying the formula on all variables (x_1 to x_n).

Step.3 Remove all $*$-edges.

For example, the sum-of-products Ternary Decision Diagram in Figure 3.13(a) is given, first we can obtain the sum-of-minterms Ternary Decision Diagram in (b), second the "irredundant" sum-of-products Ternary Decision Diagram in (c), and last the Binary Decision Diagram in (d). The SOP-TDD in (a) represents $\overline{x_1}\,\overline{x_2}+x_1x_2+\overline{x_3}$, the SOM-TDD in (b) represents $\overline{x_1}\,\overline{x_2}\,\overline{x_3}+\overline{x_1}\,\overline{x_2}x_3+\overline{x_1}x_2\overline{x_3}+x_1\overline{x_2}\,\overline{x_3}+x_1x_2\overline{x_3}+x_1x_2x_3$, the TDD in (c) and the BDD in (d) represent $\overline{x_1}x_2\overline{x_3}+\overline{x_1}\,\overline{x_2}+x_1\overline{x_2}\,\overline{x_3}+x_1x_2$. They all represent the same Boolean function.

We emphasize here that the conversions, especially steps 1 and 2, can be realized within the procedures for sum-of-products Ternary Decision Diagrams.

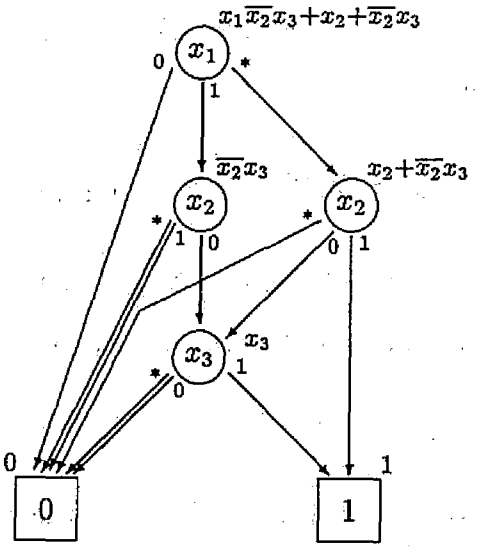
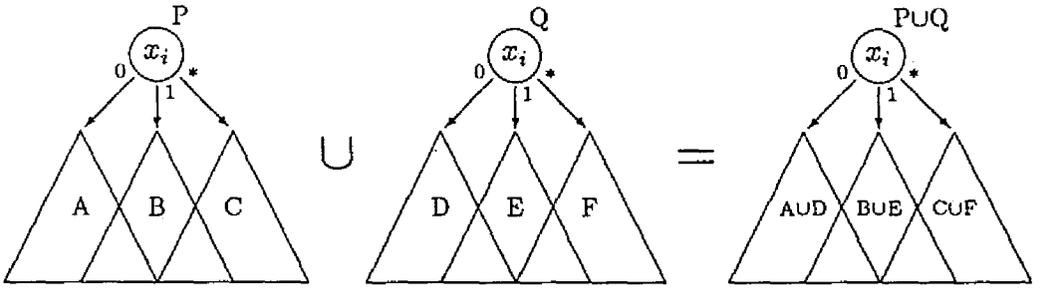
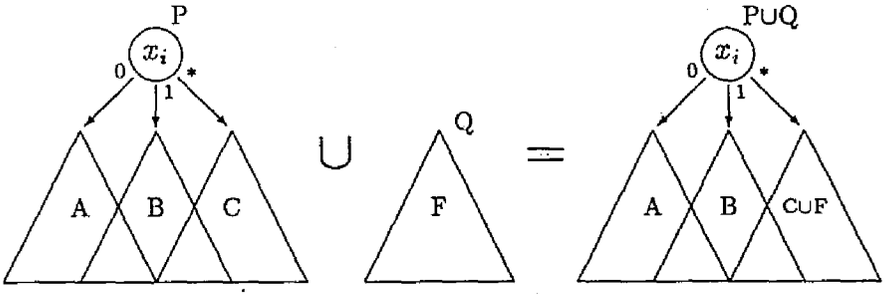


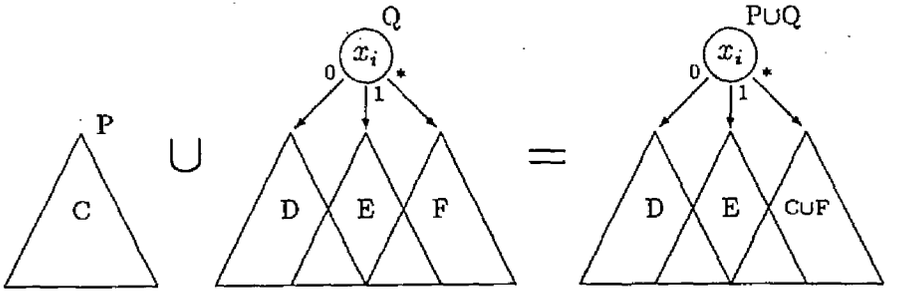
Figure 3.1: Sum-of-products forms represented in TDDs



(a) λ(node of P)=λ(node of Q)

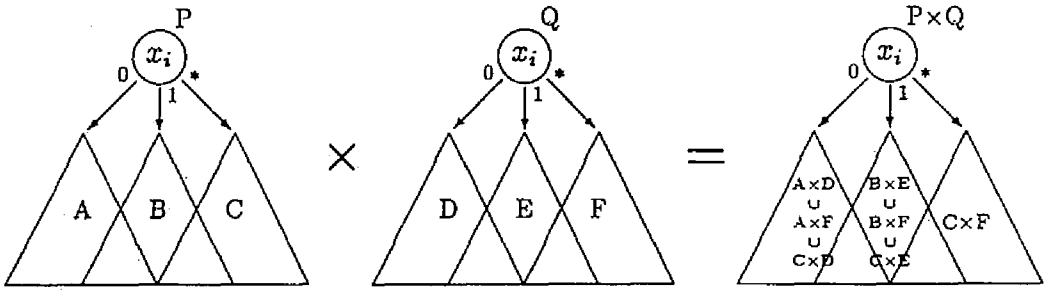


(b) λ(node of P)>λ(node of Q)

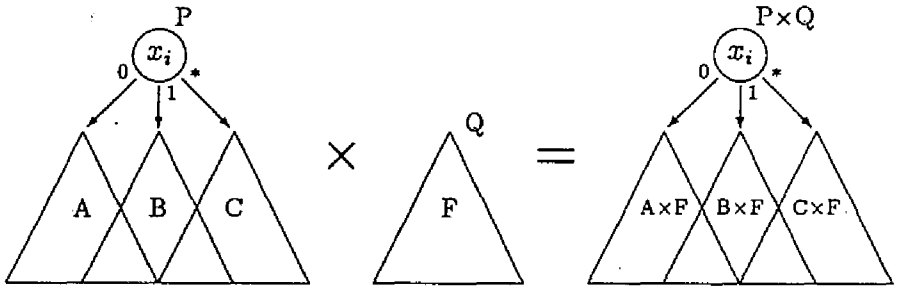


(c) λ(node of P)<λ(node of Q)

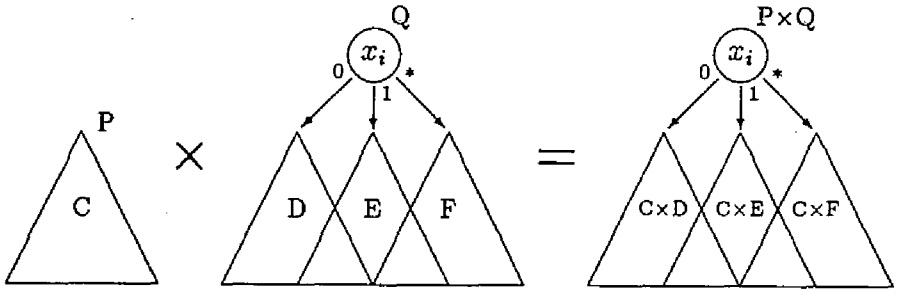
Figure 3.2: Procedure to compute PUQ



(a) λ(node of P)=λ(node of Q)

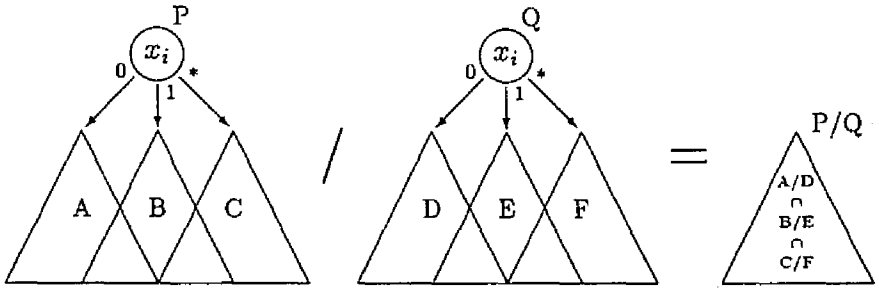


(b) λ(node of P)>λ(node of Q)

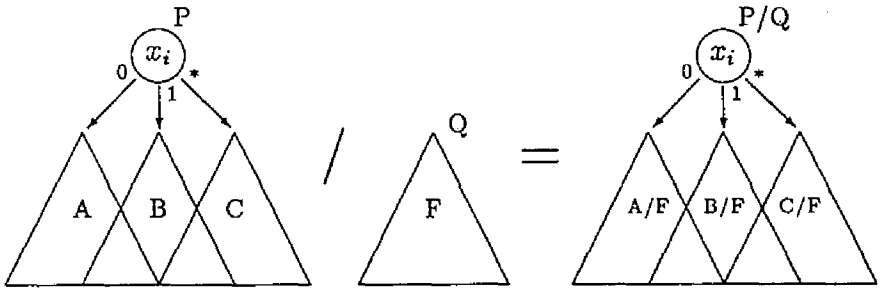


(c) λ(node of P)<λ(node of Q)

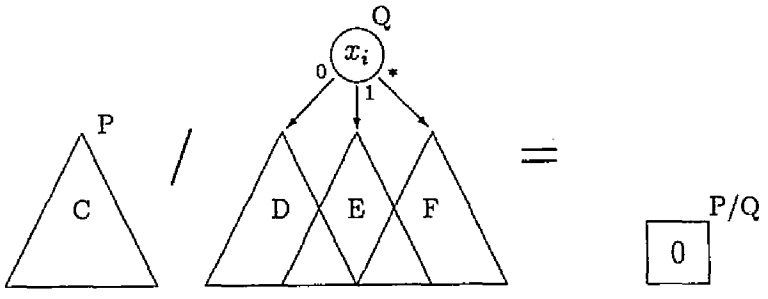
Figure 3.3: Procedure to compute P×Q



(a) λ(node of P)=λ(node of Q)

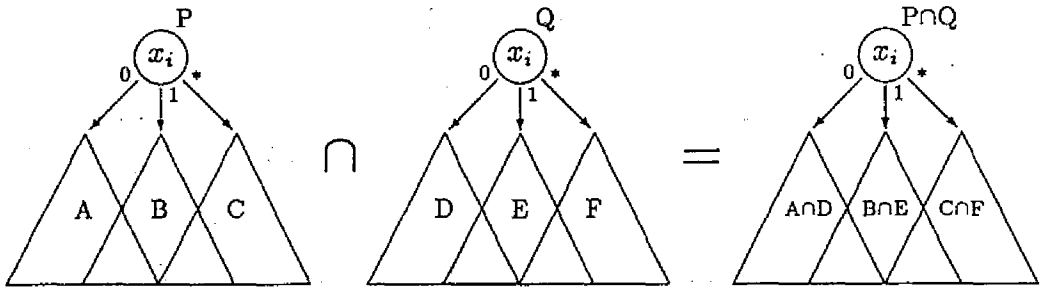


(b) λ(node of P)>λ(node of Q)

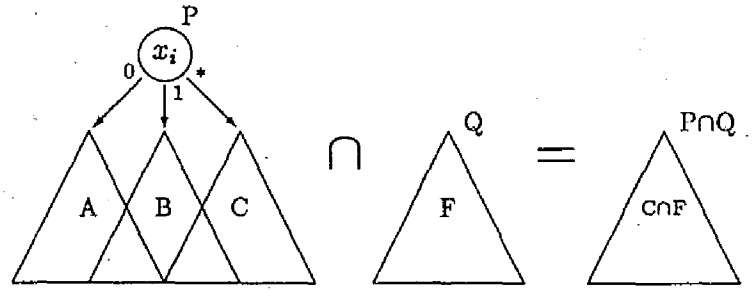


(c) λ(node of P)<λ(node of Q)

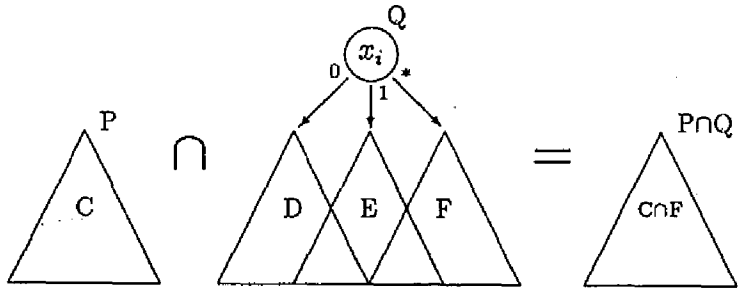
Figure 3.4: Procedure to compute P/Q



(a) λ(node of P)=λ(node of Q)

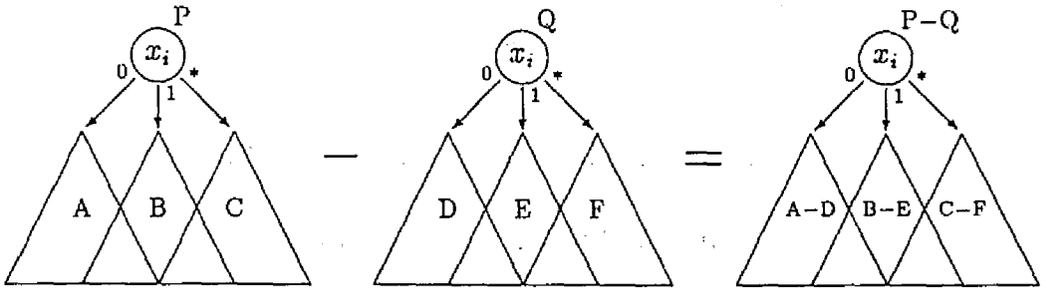


(b) λ(node of P)>λ(node of Q)

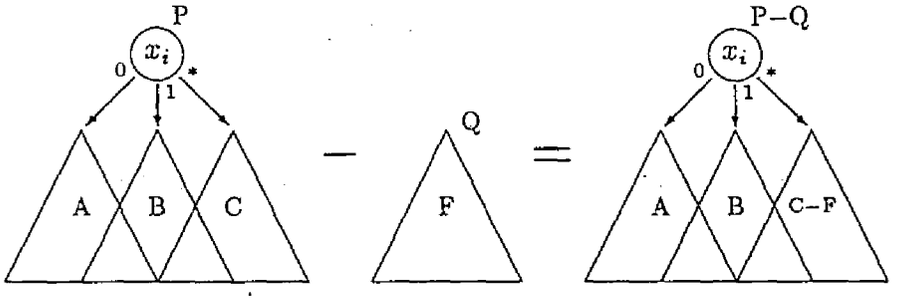


(c) λ(node of P)<λ(node of Q)

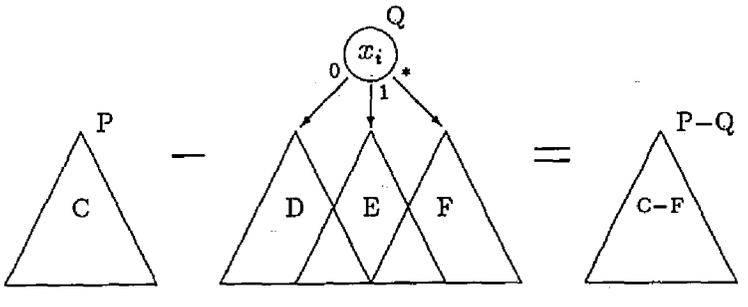
Figure 3.5: Procedure to compute P∩Q



(a) λ(node of P)=λ(node of Q)



(b) λ(node of P)>λ(node of Q)



(c) λ(node of P)<λ(node of Q)

Figure 3.6: Procedure to compute P−Q

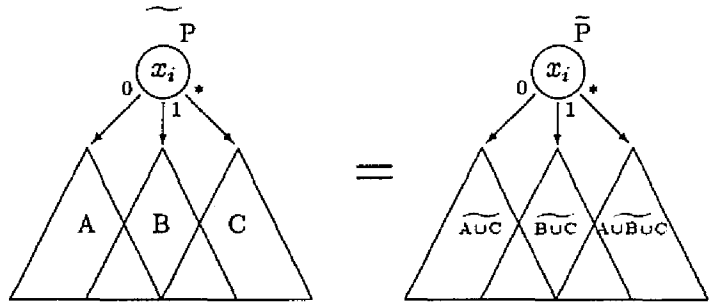


Figure 3.7: Procedure to compute $\tilde{P}$

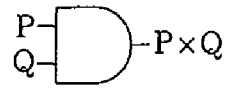


Figure 3.8: AND-gate manipulation on SOP-TDDs

Figure 3.9: OR-gate manipulation on SOP-TDDs

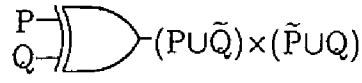


Figure 3.10: XOR-gate manipulation on SOP-TDDs

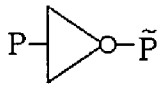
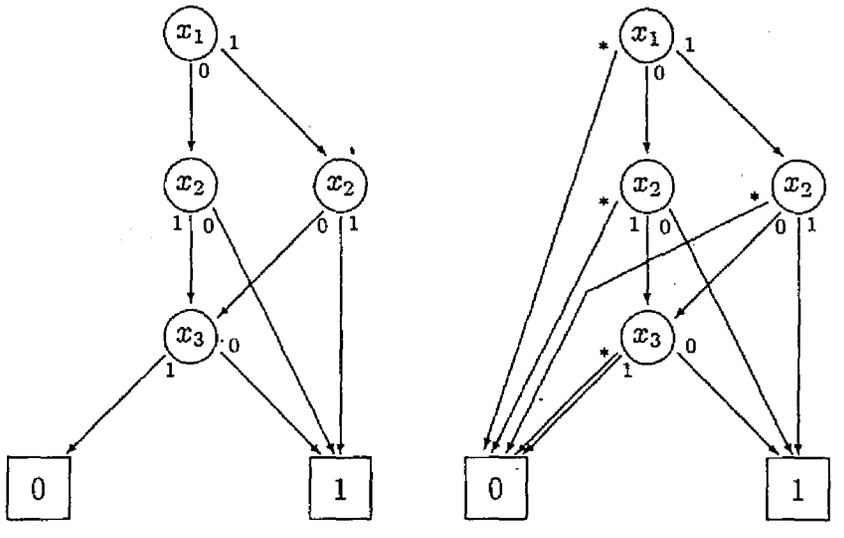
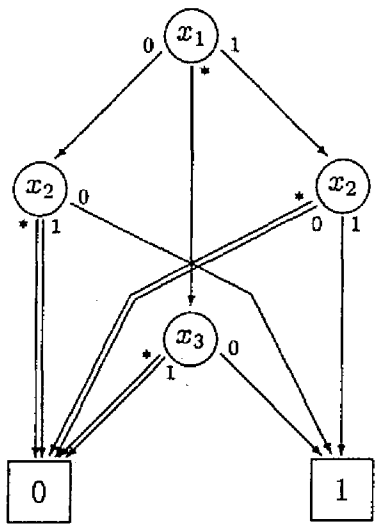


Figure 3.11: NOT-gate manipulation on SOP-TDDs

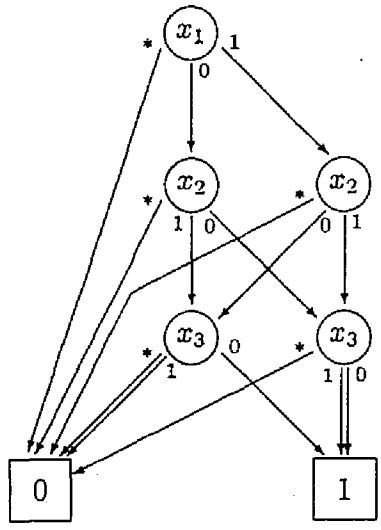


(a) given BDD (b) SOP-TDD

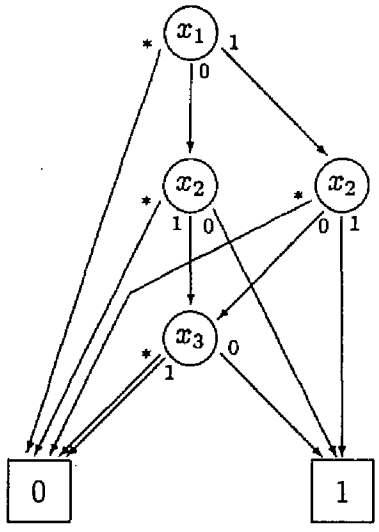
Figure 3.12: Convert BDD into SOP-TDD



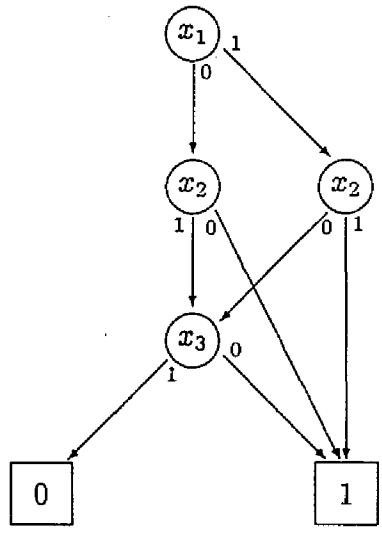
(a) given SOP-TDD



(b) SOM-TDD



(c) "irredundant" SOP-TDD



(d) BDD

Figure 3.13: Convert SOP-TDD into BDD

Chapter 4

Ringsum-of-Products Operations on TDDs

4.1 Introduction

When a set of products is given, it can be regarded either as a ringsum-of-products form or as a sum-of-products form. In this chapter, we regard that Ternary Decision Diagrams represent ringsum-of-products forms by sets of products. In other words, all TDDs appearing in this chapter are for ringsum-of-products forms, and all set operations are defined as they are used on operations on ringsum-of-products forms. Figure 4.1 shows an example of ringsum-of-products forms represented by TDD.

We conjecture all procedures mentioned in this chapter have worst case time complexity $O(|R|)$ where $|R|$ is the worst size of resulting TDDs, assuming that the operation at each node requires constant time and that the same operation is never computed twice using the operation-result table [16].

4.2 Ringsum-of-Products Operations

Here we consider essential operations on TDDs in which we represent ringsum-of-products forms. In other words, we think about "addition," "multiplication," and "logical not" in the field of ringsum-of-products forms.

4.2.1 Addition of ROP Forms

In this section, we consider "addition" of two ringsum-of-products forms. Namely we consider how to realize, for example, "$x_2 \oplus \overline{x_1} x_3$ added to $x_1 \overline{x_2} x_3 \oplus x_2 \oplus \overline{x_2} x_3$ gives $x_1 \overline{x_2} x_3 \oplus \overline{x_2} x_3 \oplus \overline{x_1} x_3$" on Ternary Decision Diagrams.

The set operator **symmetric difference**, denoted by $\oplus$, can actualize the addition of two ringsum-of-products forms. For example, $\{x_1\overline{x_2}x_3,\ x_2,\ \overline{x_2}x_3\} \oplus \{x_2,\ \overline{x_1}x_3\} = \{x_1\overline{x_2}x_3,\ \overline{x_2}x_3,\ \overline{x_1}x_3\}$ can be regarded as $(x_1\overline{x_2}x_3 \oplus x_2 \oplus \overline{x_2}x_3) \oplus (x_2 \oplus \overline{x_1}x_3) = x_1\overline{x_2}x_3 \oplus \overline{x_2}x_3 \oplus \overline{x_1}x_3$. We can realize the procedure for the operator $\oplus$ on TDDs as shown in Figure 4.2. The procedure is mainly based on the following formula:

$$(\overline{x_i}\mathrm{A} \oplus x_i\mathrm{B} \oplus \mathrm{C}) \oplus (\overline{x_i}\mathrm{D} \oplus x_i\mathrm{E} \oplus \mathrm{F}) = \overline{x_i}(\mathrm{A} \oplus \mathrm{D}) \oplus x_i(\mathrm{B} \oplus \mathrm{E}) \oplus (\mathrm{C} \oplus \mathrm{F})$$

and it terminates when it reaches to the leaf nodes, where the rules $\mathrm{P} \oplus \emptyset = \mathrm{P}$, $\emptyset \oplus \mathrm{Q} = \mathrm{Q}$, and $\{1\} \oplus \{1\} = \emptyset$ are used.

4.2.2 Multiplication of ROP Forms

Here we consider "multiplication" of two ringsum-of-products forms. Namely we consider how to realize, for example, "$x_1\overline{x_2}x_3 \oplus x_2 \oplus \overline{x_2}x_3$ multiplied by $x_1 \oplus \overline{x_3}$ gives $x_1x_2 \oplus x_2\overline{x_3}$" on Ternary Decision Diagrams.

We define a set operator **ringsum product**, denoted by $\otimes$, to actualize the multiplication of two ringsum-of-products forms. Namely, we define the operator $\otimes$ which satisfies, for example, $\{x_1\overline{x_2}x_3,\ x_2,\ \overline{x_2}x_3\} \otimes \{x_1,\ \overline{x_3}\} = \{x_1x_2,\ x_2\overline{x_3}\}$. This example can be regarded as $(x_1\overline{x_2}x_3 \oplus x_2 \oplus \overline{x_2}x_3)(x_1 \oplus \overline{x_3}) = x_1x_2 \oplus x_2\overline{x_3}$ from the ringsum-of-products form

point of view. The procedure for the operator $\otimes$ on TDDs is similar to the one for $\times$ save that $\oplus$ is used to sum up three internal products instead of $\cup$ as shown in Figure 4.3. The procedure is based on the following formula:

$$(\overline{x_i}\mathrm{A}\oplus x_i\mathrm{B}\oplus\mathrm{C})\otimes(\overline{x_i}\mathrm{D}\oplus x_i\mathrm{E}\oplus\mathrm{F})=$$
$$\overline{x_i}((\mathrm{A}\otimes\mathrm{D})\oplus(\mathrm{A}\otimes\mathrm{F})\oplus(\mathrm{C}\otimes\mathrm{D}))\oplus x_i((\mathrm{B}\otimes\mathrm{E})\oplus(\mathrm{B}\otimes\mathrm{F})\oplus(\mathrm{C}\otimes\mathrm{E}))\oplus(\mathrm{C}\otimes\mathrm{F})$$

and it terminates when it reaches to the leaf nodes, where the rules $\mathrm{P}\otimes\emptyset=\emptyset$ and $\mathrm{P}\otimes\{1\}=\mathrm{P}$ are used.

Then we consider an inverse operation for the multiplication of ringsum-of-products forms. In other words, we consider "division" on the field of ringsum-of-products forms.

In order to make the "division" one-valued, we define a set operation named **ringsum weak division**, denoted by $\oslash$, as described below:

> When sets of products P and Q are given, the quotient $\mathrm{R}=\mathrm{P}\oslash\mathrm{Q}$ is the maximum set that satisfies $\mathrm{R}\otimes\mathrm{Q}\subseteq\mathrm{P}$, where R does not include variables appeared in Q.

For example, $\{x_1\overline{x_2}x_3,\ x_2,\ \overline{x_2}x_3\}\oslash\{1,\ x_1\}=\{\overline{x_2}x_3\}$. In the fact, however, this operator $\oslash$ is the same as the operator / mentioned in the previous chapter, since $\mathrm{R}\otimes\mathrm{Q}$ is equal to $\mathrm{R}\times\mathrm{Q}$ when R does not include variables appeared in Q. Thus in the rest of this thesis we use the weak division operator / for the "division" of ringsum-of-products form. Figure 4.4 shows the procedure for the operator / on TDDs. The procedure terminates when the rule $\mathrm{P}/\emptyset=\infty$ or $\mathrm{P}/\{1\}=\mathrm{P}$ is used, where ∞ means the universal set which satisfies $\infty\cap\mathrm{R}=\mathrm{R}$ for any set R.

Lastly, we consider getting a remainder after a weak division on the field of ringsum-of-products forms. In other words, after $x_1\overline{x_2}x_3\oplus x_2\oplus\overline{x_2}x_3$ divided by $1\oplus x_1$ gives the quotient $\overline{x_2}x_3$, we should think how to get the remainder x_2.

The set operator symmetric difference $\oplus$, mentioned in the previous section, is suitable for the purpose since we can get the remainder with $\mathrm{P}\oplus((\mathrm{P}/\mathrm{Q})\otimes\mathrm{Q})$ as a set of products. The procedure for the operator $\oplus$ on TDDs is shown in Figure 4.2.

4.2.3 Logical Not of ROP Forms

When a ringsum-of-products form F is given, we often need a "logical not" ringsum-of-products form F′ of F which satisfies:

- F added to F′ gives a ringsum-of-products form of the Boolean function 1, and
- F multiplied by F′ gives a ringsum-of-products form of the Boolean function 0.

Here we define an unary set operation **ringsum complement**, denoted by $\widehat{\ }$, to get one of such "logical not" ringsum-of-products form which is represented by a set of products. Figure 4.5 shows a procedure for the operator $\widehat{\ }$ on TDDs. The procedure is based on the following formulae:

$$(\overline{x_i}\mathrm{A}\oplus\widehat{x_i}\mathrm{B}\oplus\mathrm{C})=\overline{x_i}\mathrm{A}\oplus x_i\mathrm{B}\oplus\widehat{\mathrm{C}}$$
$$(\overline{x_i}\oplus\widehat{x_i}\mathrm{B}\oplus\mathrm{C})=x_i\widehat{\mathrm{B}}\oplus\mathrm{C}$$
$$(\overline{x_i}\mathrm{A}\widehat{\oplus x_i}\oplus\mathrm{C})=\overline{x_i}\widehat{\mathrm{A}}\oplus\mathrm{C}$$

and it terminates when it reaches to the leaf nodes, where the rules $\widehat{\emptyset}=\{1\}$ and $\widehat{\{1\}}=\emptyset$ are used.

4.3 Combinatorial Circuits Manipulation Using ROP-TDDs

In this section, we mention about the method to generate representation of outputs of all gates in combinatorial circuits under the ringsum-of-products Ternary Decision Diagrams. Here we assume that the circuits consist of 2-input AND-gates, 2-input OR-gates, 2-input XOR-gates, and 1-input NOT-gates. We also assume that primary inputs are represented by given sets of $\{x_i\}$. So in this section,

i) **2-input AND-gate**

When a 2-input AND-gate, whose inputs are represented in ringsum-of-products forms

P and Q, is given, we generate P⊗Q for its output as shown in Figure 4.6. For example, when the two inputs of an AND-gate are represented in $x_1\overline{x_2}x_3 \oplus x_2 \oplus \overline{x_2}x_3$ and $x_1 \oplus \overline{x_3}$, its output results in $x_1x_2 \oplus x_2\overline{x_3}$. TDDs actualize this operation as the ringsum product procedure mentioned in the previous section.

ii) **2-input OR-gate**

When a 2-input OR-gate, whose inputs are represented in ringsum-of-products forms P and Q, is given, we generate $\widehat{P \otimes Q}$ for one of the ringsum-of-products forms of its output as shown in Figure 4.7. For example, when the two inputs of an OR-gate are represented in $x_1\overline{x_2}x_3 \oplus x_2 \oplus \overline{x_2}x_3$ and $x_1 \oplus \overline{x_3}$ its output results in $\overline{x_1}\,\overline{x_2}\,\overline{x_3} \oplus \overline{x_2}\,\overline{x_3} \oplus 1$. TDDs actualize this operation as the combination of the ringsum complement procedure and the ringsum product procedure mentioned in the previous section.

iii) **2-input XOR-gate**

When a 2-input XOR-gate, whose inputs are represented in ringsum-of-products forms P and Q, is given, we generate P⊕Q for its output as shown in Figure 4.8. For example, when the two inputs of an XOR-gate are represented in $x_1\overline{x_2}x_3 \oplus x_2 \oplus \overline{x_2}x_3$ and $x_2 \oplus \overline{x_1}x_3$ its output results in $x_1\overline{x_2}x_3 \oplus \overline{x_2}x_3 \oplus \overline{x_1}x_3$. TDDs actualize this operation as the symmetric difference procedure mentioned in the previous section.

iv) **NOT-gate**

When a NOT-gate, whose input is represented in ringsum-of-products form P, is given, we generate $\widehat{P}$ for its output as shown in Figure 4.9. For example, when the input of a NOT-gate is represented in $x_1\overline{x_2}x_3 \oplus x_2 \oplus \overline{x_2}x_3$, its output results in $x_1\overline{x_2}x_3 \oplus \overline{x_2}\,\overline{x_3}$. TDDs actualize this operation as the ringsum complement procedure mentioned in the previous section.

4.4 Conversion between ROP-TDDs and BDDs

In this section, we consider a method to convert a Binary Decision Diagram into a ringsum-of-products Ternary Decision Diagram which represents the same Boolean function represented by the BDD, vice versa.

When a Binary Decision Diagram is given, we can obtain a ringsum-of-products Ternary Decision Diagram which represents the same Boolean function represented by the given BDD, adding $\boxed{0}$-directed $*$-edges to all non-leaf nodes of the BDD. For example, the Binary Decision Diagram in Figure 4.10(a) is given, we can obtain the Ternary Decision Diagram in (b). They both represent the same Boolean function $\overline{x_1}x_2\overline{x_3} \oplus \overline{x_1}\,\overline{x_2} \oplus x_1\overline{x_2}\,\overline{x_3} \oplus x_1x_2$. This is because all the products, which we can derive from all paths from the top node to $\boxed{1}$ in a Binary Decision Diagram, are disjointed with one another. Therefore, the Boolean function represented by the BDD is regarded as the ringsum of the products, and it is easily represented by the ROP-TDD whose 0-edges and 1-edges constitutes the same graph of the BDD and whose $*$-edges direct $\boxed{0}$.

Inversely, when a ringsum-of-products Ternary Decision Diagram is given, we can obtain a Binary Decision Diagram which represents the same Boolean function represented by the given ROP-TDD, as follows:

Step.1 Generate Ternary Decision Diagram which represents the set of minterms, where the ringsum of the minterms expresses the same Boolean function as the given ROP-TDD. In order to generate such "ringsum-of-minterms" TDD Q from the given ROP-TDD P, we use Shannon-Davio Expansion as described below:

$$\mathrm{Q} = \{\overline{x_1}, x_1\} \otimes \{\overline{x_2}, x_2\} \otimes \cdots \otimes \{\overline{x_n}, x_n\} \otimes \mathrm{P} \ .$$

Step.2 Remove nodes whose 0-edge and 1-edge point the same node. In order to remove x_i-labeled ones of such "redundant" nodes from the Ternary Decision

Diagram Q and obtain the new Ternary Decision Diagram Q', we use the following formula:

$$Q' = Q \oplus (Q/\{\overline{x_i}, x_i\} \otimes \{\overline{x_i}, x_i, 1\}) .$$

We can remove all nodes whose 0-edge and 1-edge point the same node by applying the formula on all variables (x_1 to x_n).

Step.3 Remove all $*$-edges.

For example, the ringsum-of-products Ternary Decision Diagram in Figure 4.11(a) is given, first we can obtain the ringsum-of-minterms Ternary Decision Diagram in (b), second the "irredundant" ringsum-of-products Ternary Decision Diagram in (c), and last the Binary Decision Diagram in (d). The ROP-TDD in (a) represents $\overline{x_1}x_3 \oplus \overline{x_2}x_3 \oplus 1$, the ROM-TDD in (b) represents $\overline{x_1}\,\overline{x_2}\,\overline{x_3} \oplus \overline{x_1}\,\overline{x_2}x_3 \oplus \overline{x_1}x_2\overline{x_3} \oplus x_1\overline{x_2}\,\overline{x_3} \oplus x_1x_2\overline{x_3} \oplus x_1x_2x_3$, the TDD in (c) and the BDD in (d) represent $\overline{x_1}x_2\overline{x_3} \oplus \overline{x_1}\,\overline{x_2} \oplus x_1\overline{x_2}\,\overline{x_3} \oplus x_1x_2$. They all represent the same Boolean function.

We emphasize here that the conversions, especially steps 1 and 2, can be realized within the procedures for ringsum-of-products Ternary Decision Diagrams.

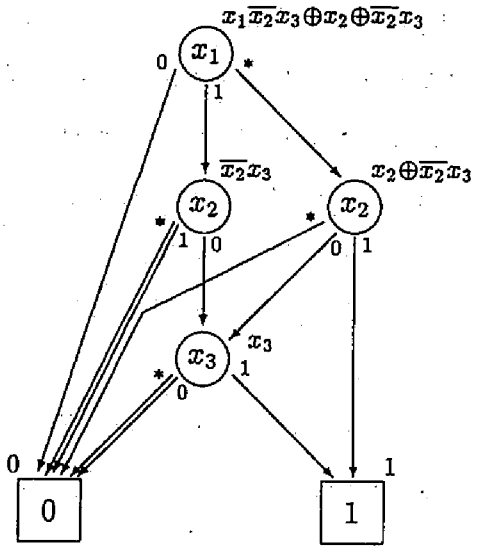
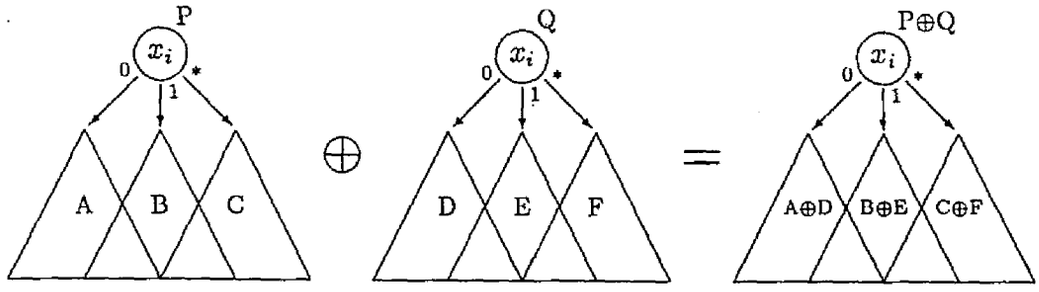
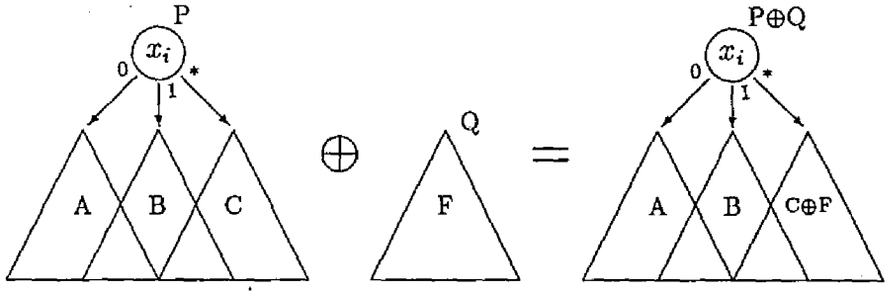


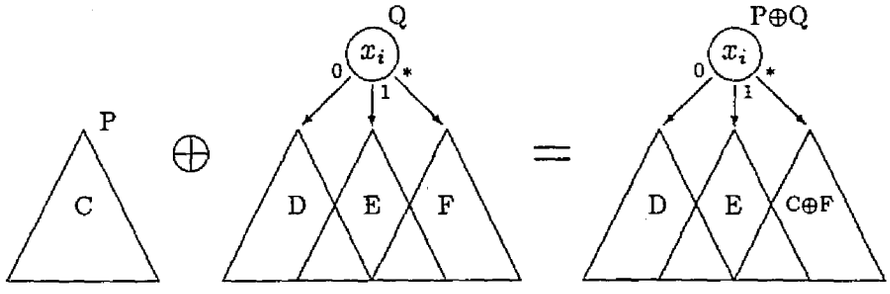
Figure 4.1: Ringsum-of-products forms represented in TDDs



(a) λ(node of P)=λ(node of Q)

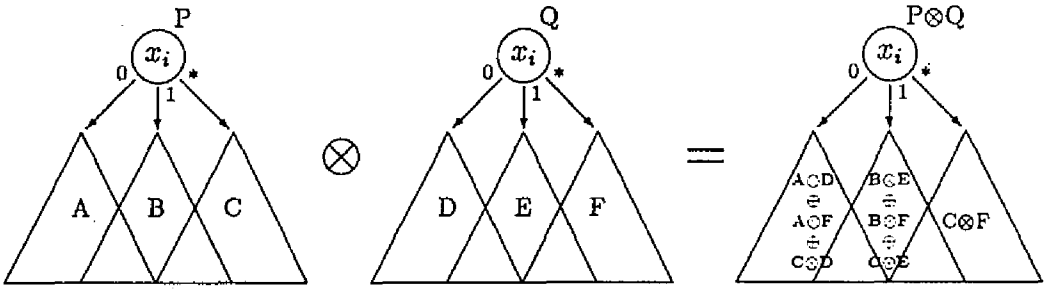


(b) λ(node of P)>λ(node of Q)

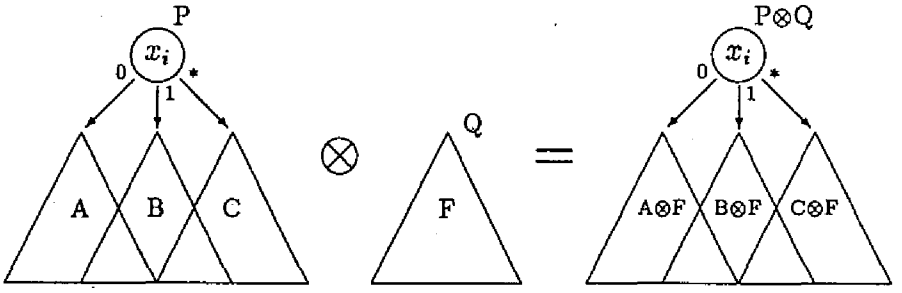


(c) λ(node of P)<λ(node of Q)

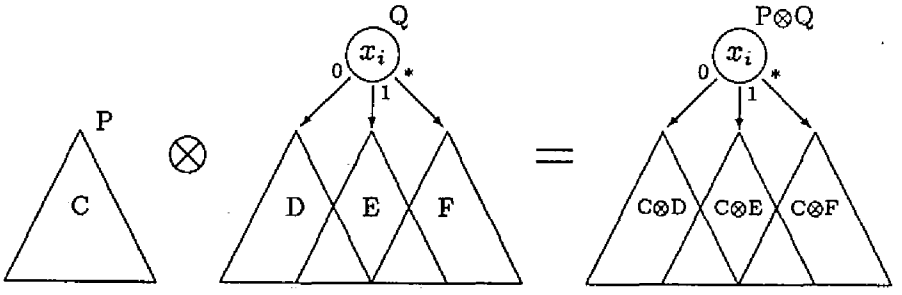
Figure 4.2: Procedure to compute P⊕Q



(a) λ(node of P)=λ(node of Q)

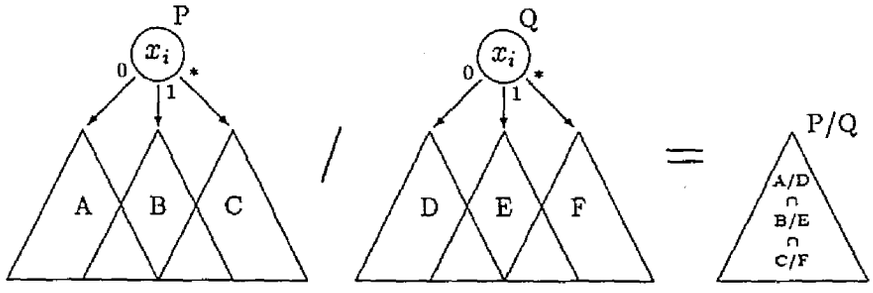


(b) λ(node of P)>λ(node of Q)

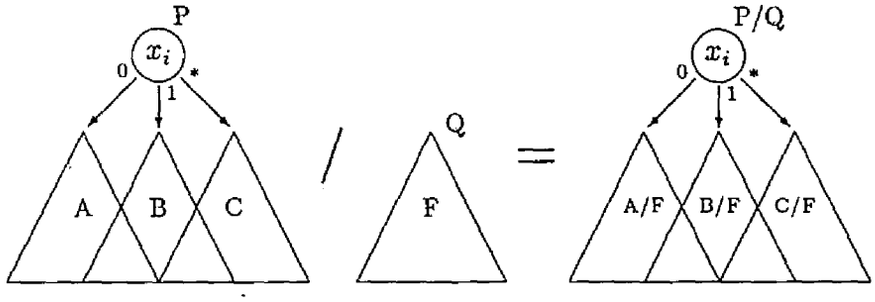


(c) λ(node of P)<λ(node of Q)

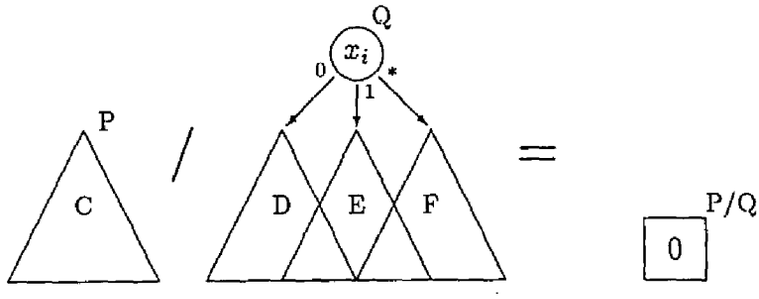
Figure 4.3: Procedure to compute P⊗Q



(a) λ(node of P)=λ(node of Q)

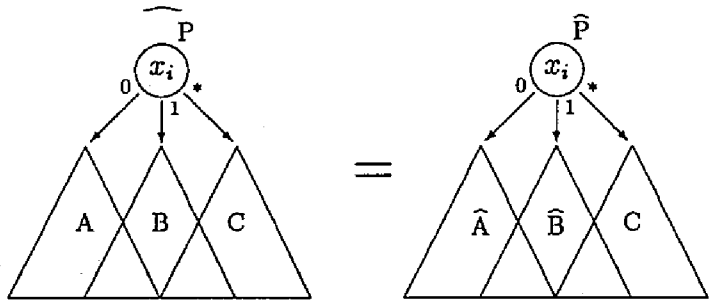


(b) λ(node of P)>λ(node of Q)

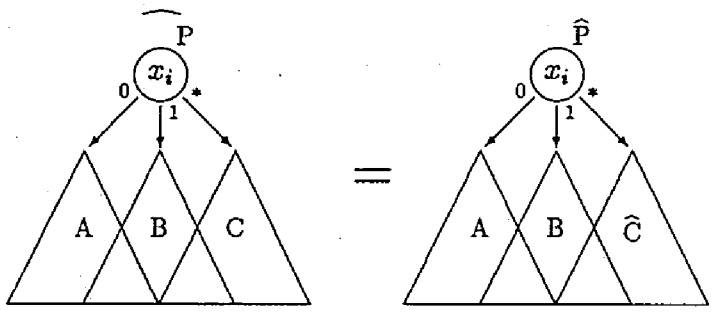


(c) λ(node of P)<λ(node of Q)

Figure 4.4: Procedure to compute P/Q



(a) A≡{1} or B≡{1}



(b) A≠{1} and B≠{1}

Figure 4.5: Procedure to compute $\widehat{P}$

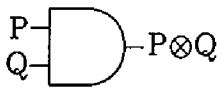


Figure 4.6: AND-gate manipulation on ROP-TDDs

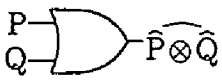


Figure 4.7: OR-gate manipulation on ROP-TDDs

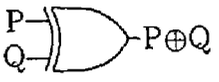


Figure 4.8: XOR-gate manipulation on ROP-TDDs

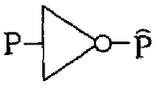
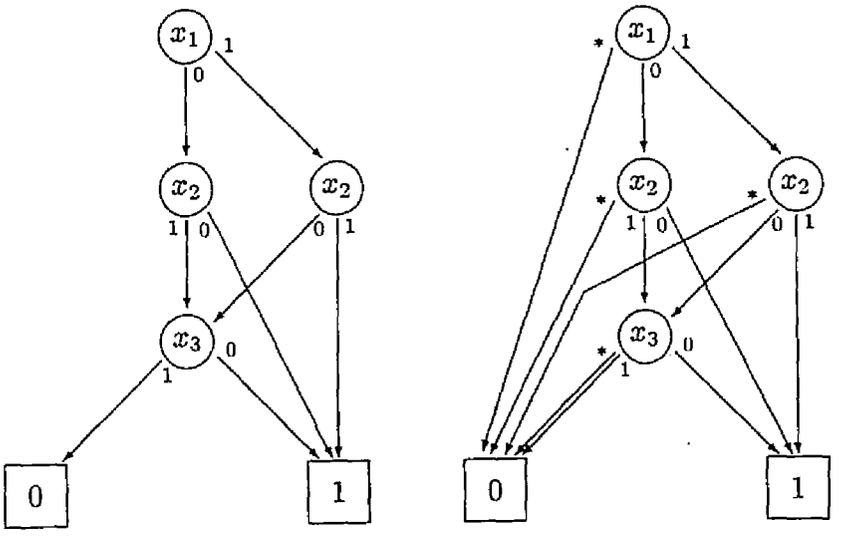
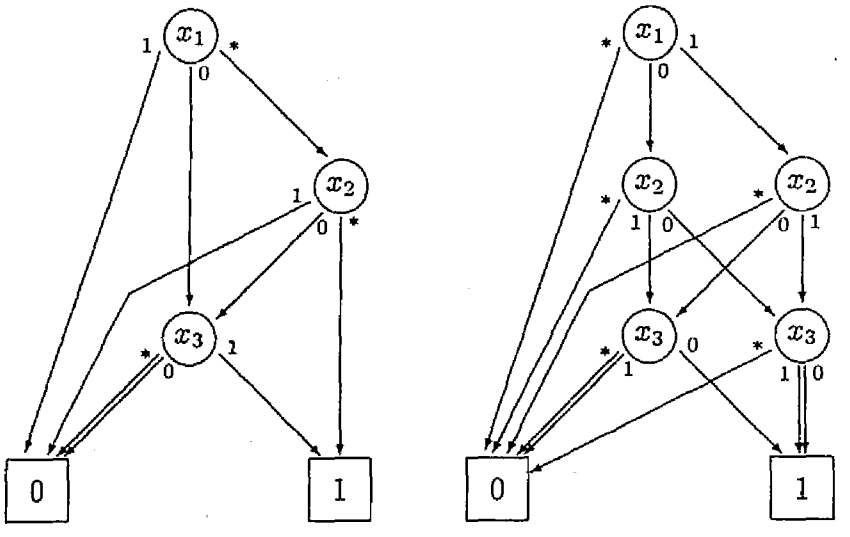


Figure 4.9: NOT-gate manipulation on ROP-TDDs

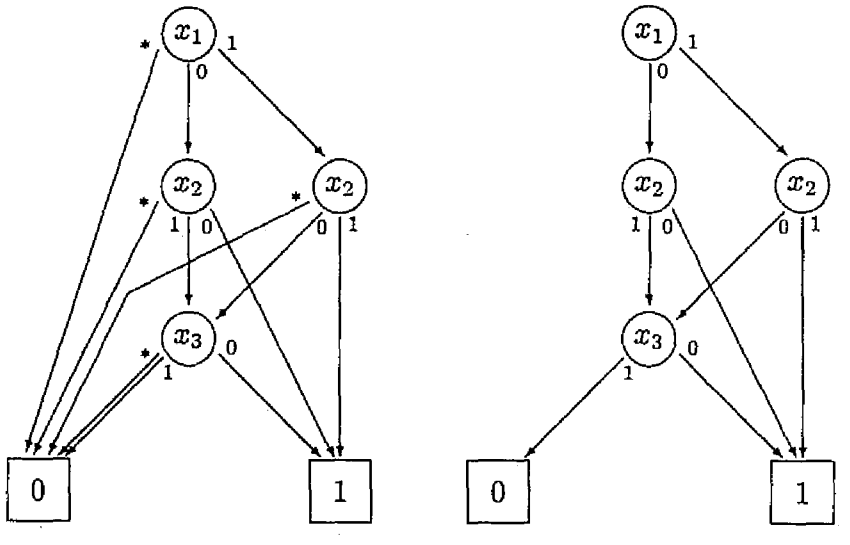


(a) given BDD (b) ROP-TDD

Figure 4.10: Convert BDD into ROP-TDD



(a) given ROP-TDD (b) ROM-TDD



(c) "irredundant" ROP-TDD (d) BDD

Figure 4.11: Convert ROP-TDD into BDD

Chapter 5

Implementation Techniques

5.1 Introduction

The procedures mentioned in chapters 3 and 4 have been implemented under UNIX C. The implemented Ternary Decision Diagram Library consists of 13 user-accessible routines, 3 internal routines, one user-accessible header file, and one internal header file. In this chapter, we discuss about the implementation of the Ternary Decision Diagram Library.

5.2 Structure of Nodes

In the Ternary Decision Diagram Library, the non-leaf node structure `struct tdd_struct`, or shortly `tdd`, is defined as follows:

```
typedef struct tdd_struct{
  struct tdd_struct *tdd_zero_edge;
  struct tdd_struct *tdd_one_edge;
  struct tdd_struct *tdd_asterisk_edge;
  int tdd_in_counter;
  int tdd_timestamp;
```

```
    unsigned int tdd_variable;
    struct tdd_struct *tdd_hashlink;
} tdd;
```

The elements `tdd_zero_edge`, `tdd_one_edge`, `tdd_asterisk_edge`, and `tdd_variable` store particular values of the 0-edge, the 1-edge, the $*$-edge, and the variable (x_i) of the non-leaf node of Ternary Decision Diagrams, respectively. The four values make a hashed-key-number for each non-leaf node, and `tdd_hashlink` is used for synonyms. `tdd_in_counter` counts the number of incoming edges to the node. When `tdd_in_counter` goes to zero, the node is considered unused and will be reused to represent another set of products. `tdd_timestamp` counts up how many times the node is reused. The functions of `tdd_in_counter` and `tdd_timestamp` are mentioned in section 5.5 in detail.

The leaf nodes $\boxed{0}$ and $\boxed{1}$ are defined as constants shown below:

```
#define TDDEMPTY ((tdd*)1)
#define TDD1SET ((tdd*)3)
```

where `TDDEMPTY` means $\emptyset$ and `TDD1SET` means $\{1\}$. The universal set ∞ is defined as

```
#define TDDUNIV ((tdd*)2)
```

for the division by $\emptyset$.

These definitions are all in the user-accessible header file named "`tdd.h`".

5.3 Structure of Operation-Result Table

In order to avoid calculating the same operation twice or more, we use **operation-result table** which stores the result of the set operations on Ternary Decision Diagrams. For example, once the TDD representing $\{\overline{x_1}\,\overline{x_2}\,\overline{x_3},\ \overline{x_1}\,\overline{x_2}x_3,\ \overline{x_1}x_2\overline{x_3},\ x_1\overline{x_2}\,\overline{x_3},\ x_1x_2\overline{x_3},\ x_1x_2x_3\}$ is constructed by the operation $\{\overline{x_1}\,\overline{x_2},\ x_1x_2{+}x_3\}\times\{\overline{x_1}\,\overline{x_2}\,\overline{x_3},\ \overline{x_1}\,\overline{x_2}x_3,\ \overline{x_1}x_2\overline{x_3},\ \overline{x_1}x_2x_3,$

$x_1\overline{x_2}\,\overline{x_3}$, $x_1\overline{x_2}x_3$, $x_1x_2\overline{x_3}$, $x_1x_2x_3\}$ by `tddprod` (mentioned in section 5.4) as shown in Figure 5.1, the construction of $\{\overline{x_1}\,\overline{x_2}\,\overline{x_3}$, $\overline{x_1}\,\overline{x_2}x_3$, $\overline{x_1}x_2\overline{x_3}$, $x_1\overline{x_2}\,\overline{x_3}$, $x_1x_2\overline{x_3}$, $x_1x_2x_3\}$ will never occur for the operation $\{\overline{x_1}\,\overline{x_2}$, $x_1x_2+x_3\}\times\{\overline{x_1}\,\overline{x_2}\,\overline{x_3}$, $\overline{x_1}\,\overline{x_2}x_3$, $\overline{x_1}x_2\overline{x_3}$, $\overline{x_1}x_2x_3$, $x_1\overline{x_2}\,\overline{x_3}$, $x_1\overline{x_2}x_3$, $x_1x_2\overline{x_3}$, $x_1x_2x_3\}$ while `tddprod` only increments the `tdd_in_counter` of the top node of the TDD down below in Figure 5.1 and returns the top node. This is because the operation-result table memorizes that

> "The result of $\{\overline{x_1}\,\overline{x_2}$, $x_1x_2+x_3\}\times\{\overline{x_1}\,\overline{x_2}\,\overline{x_3}$, $\overline{x_1}\,\overline{x_2}x_3$, $\overline{x_1}x_2\overline{x_3}$, $\overline{x_1}x_2x_3$, $x_1\overline{x_2}\,\overline{x_3}$, $x_1\overline{x_2}x_3$, $x_1x_2\overline{x_3}$, $x_1x_2x_3\}$ is $\{\overline{x_1}\,\overline{x_2}\,\overline{x_3}$, $\overline{x_1}\,\overline{x_2}x_3$, $\overline{x_1}x_2\overline{x_3}$, $x_1\overline{x_2}\,\overline{x_3}$, $x_1x_2\overline{x_3}$, $x_1x_2x_3\}$."

In the Ternary Decision Diagram Library, the operation-result table structure `struct tdd_calc_tables_struct` is defined as follows:

```
struct tdd_calc_tables_struct{
  tdd *tdd_left_arg;
  int tdd_left_timestamp;
  tdd *tdd_right_arg;
  int tdd_right_timestamp;
  tdd *tdd_answer_arg;
  int tdd_answer_timestamp;
  tdd **tdd_calc_identifier;
};
```

The elements `tdd_left_arg` and `tdd_right_arg` point the argument nodes of operations, and `tdd_left_timestamp` and `tdd_right_timestamp` store their `tdd_timestamps`. `tdd_calc_identifier` stores the proper value to distinguish the set operations, namely union, Cartesian product, weak division, intersection, difference, complement, symmetric difference, ringsum product, and ringsum complement. `tdd_answer_arg` points the result node of the operation, and `tdd_answer_timestamp` stores its `tdd_timestamp`. This definition is in the internal header file named "`tdd_internal.h`".

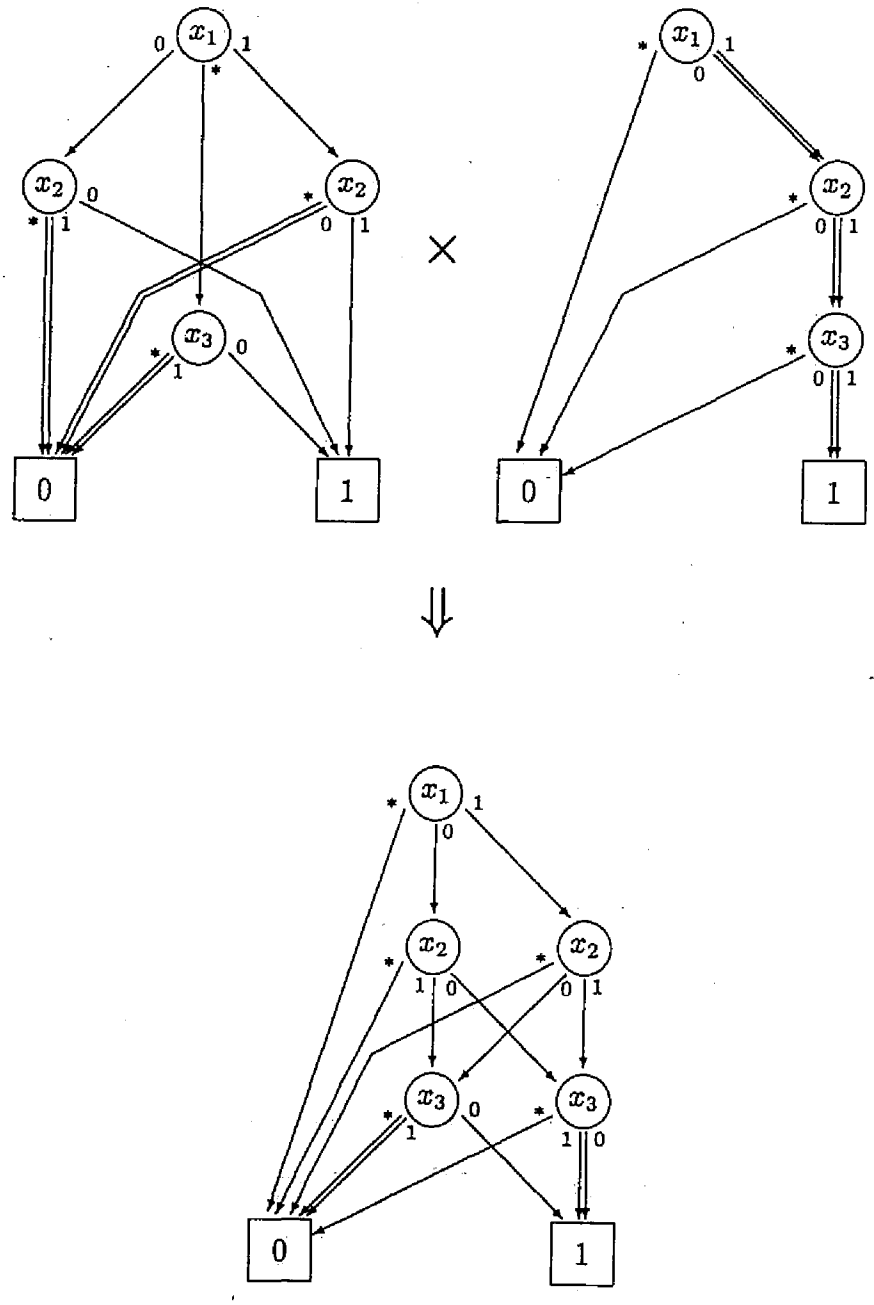


Figure 5.1: Same Operation Never Occurs Twice

5.4 User-Accessible Routines

To obtain basic sets $\{\overline{x_i}\}$ and $\{x_i\}$, the Ternary Decision Diagram Library has two routines:

```
tdd *tddnlit(unsigned int i)
tdd *tddplit(unsigned int i)
```

`tddnlit` returns the set $\{\overline{x_i}\}$ and `tddplit` returns $\{x_i\}$.

To increment and decrement `tdd_in_counter` of a non-leaf node, we use

```
tdd *tdddup(tdd *p)
int tddfree(tdd *p)
```

respectively. `tdddup` returns the argument itself, and `tddfree` returns the error number. `tdddup` is considered as the duplication of the non-leaf node, and `tddfree` as the disposal. `tdddup` and `tddfree` do nothing when the argument is $\emptyset$ or $\{1\}$.

The Ternary Decision Diagram Library includes nine routines for the nine set operations, namely union, Cartesian product, weak division, intersection, difference, complement, symmetric difference, ringsum product, and ringsum complement:

```
tdd *tddunion(tdd *p,tdd *q)
tdd *tddprod(tdd *p,tdd *q)
tdd *tddwdiv(tdd *p,tdd *q)
tdd *tddinter(tdd *p,tdd *q)
tdd *tdddiff(tdd *p,tdd *q)
tdd *tddcmpl(tdd *p)
tdd *tddsdiff(tdd *p,tdd *q)
tdd *tddrprod(tdd *p,tdd *q)
tdd *tddrcmpl(tdd *p)
```

All these routines are implemented along the procedures mentioned in chapters 3 and 4, and they use the operation-result table to avoid calculating the same operation twice or more. For example, we discuss about the case when `tddunion` is called to obtain PUQ.

Step.1 "Normalize" the operation for P∪Q. Here "normalize" means that P and Q are swapped when Q has a smaller hashed-key-number than P. Of course, `tddwdiv` and `tdddiff` do not "normalize" their operations, and `tddcmpl` and `tddrcmpl` cannot do.

Step.2 Check the termination of the procedure. In other words, check whether P is equal to $\emptyset$ or whether Q is equal to $\{1\}$. ($\emptyset$ has the smallest hashed-key-number and $\{1\}$ has the second smallest. They have no synonyms.) If so, call `tdddup` with Q to return it as the result.

Step.3 Check the operation-result table for the result of P∪Q. When found, call `tdddup` with the result to return it.

Step.4 Perform the calculation along the procedure mentioned in chapter 3. Namely, call `tddunion` three times for the operations of 0-edges, 1-edges, and $*$-edges, then call `tdd_make_node` (mentioned in section 5.5) to make a new node R whose edges point the result of the above three `tddunions`.

Step.5 Register "The result of P∪Q is R" on the operation-result table. Also register "The result of P−R is $\emptyset$" and "The result of Q−R is $\emptyset$" on the operation-result table. Then return R as the result.

In `tddsdiff`, when P⊕Q=R is registered on the operation-result table, Q⊕R=P and P⊕R=Q are also registered. In `tddcmpl` the registration of $\tilde{P}$=Q is followed by the registration of P×Q=$\emptyset$. Also in `tddrcmpl` $\hat{P}$=Q by P⊗Q=$\emptyset$.

5.5 Internal Routines

The Ternary Decision Diagram Library has three internal routines:

```
tdd *tdd_make_node(tdd *p,tdd *q,tdd *r,unsigned int i)
void tdd_calc_reg(tdd *p,tdd *q,tdd **c,tdd *r)
tdd *tdd_calc_find(tdd *p,tdd *q,tdd **c)
```

`tdd_make_node` makes the non-leaf node whose 0-edge, 1-edge, $*$-edge, and variable are `p`, `q`, `r`, and x_i, respectively. If such node already exists, `tdd_make_node` only calls `tdddup` with the node to satisfy Node Unification Rule mentioned in chapter 2. Otherwise, `tdd_make_node` finds an unused node, whose `tdd_in_counter` is equal to 0, from the hashed node table and use it for the new node with incrementing its `tdd_timestamp`. If the unused node is a reused node, whose `tdd_timestamp` was not 0, `tdd_make_node` calls `tddfree` to dispose the nodes pointed by its 0-edge, 1-edge, and $*$-edge before the reuse.

In order to satisfy Node Reduction Rule mentioned in chapter 2, `tdd_make_node` first checks whether both `p` and `q` are equal to $\emptyset$ before the procedures mentioned above. If so, `tdd_make_node` only calls `tdddup` with `r`.

`tdd_calc_reg` registers "The result of p*op*q is `r`" on the operation-result table, where `c` indicates a proper value for the operation *op*. Inversely, `tdd_calc_find` returns "The result of p*op*q" when the operation is formerly performanced.

5.6 Experimental Results

In order to check out the performance of the implemented Ternary Decision Diagram Library, the sizes (i.e. the numbers of non-leaf nodes) of Ternary Decision Diagrams and Binary Decision Diagrams have been compared using several circuits from IWLS'93 Benchmark Set [27].

Table 5.1 shows the numbers of non-leaf nodes to represent the outputs of the circuits used in Ternary Decision Diagrams or Binary Decision Diagrams, when they are made straightly along the circuit descriptions in the Benchmark Set. The sizes of quasi-reduced Binary Decision Diagrams are shown for reference. Any of Ternary Decision Diagrams and

Table 5.1: Sizes of TDDs and BDDs

circuits	TDDs		QBDDs	BDDs
	SOP	ROP		
b1	8	8	11	8
b9	329	241	1505	277
c8	270	143	627	219
cc	106	85	325	102
cu	119	70	239	138
i1	47	66	228	64
i2	268	282	599	273
i3	132	132	743	132
i4	204	780	3643	420
i5	797	1230	10894	1026
i6	151	148	8532	222
i7	217	210	17547	485
i8	3463	1458	68621	5619
i9	402	370	11444	1159
x1	667	1548	7025	2004
x2	39	49	66	44
x3	3824	1060	25636	2433
x4	707	566	10142	944

Binary Decision Diagrams has been made within 10 second CPU time on a SiliconGraphics INDY Workstation.

When representing sum-of-products forms, 12 out of 18 Ternary Decision Diagrams are smaller than their corresponding Binary Decision Diagrams, and the size of Ternary Decision Diagrams is 83% of the size of Binary Decision Diagrams on the average. When representing ringsum-of-products forms, 11 out of 18 Ternary Decision Diagrams are smaller than their corresponding Binary Decision Diagrams, and the size of Ternary Decision Diagrams is 85% of the size of Binary Decision Diagrams on the average. We need 28 bytes/node for Ternary Decision Diagrams and 24 bytes/node for Binary Decision Diagrams on the implementation here, so we can say that Ternary Decision Diagrams representing sum-of-products forms are smaller than Binary Decision Diagrams and that Ternary Decision Diagrams representing ringsum-of-products forms are not larger than Binary Decision Diagrams, even when we do not optimize the sizes of forms and do not care about the variable orderings on Ternary Decision Diagrams.

Chapter 6

Applications for Logic Synthesis

6.1 Generating Prime Implicants by TDDs

Prime implicants [7] are the implicants of a given Boolean function, which do not subsume any other implicants. The prime implicants are used to obtain the minimum sum-of-products of the given Boolean function. In this section, we discuss about the method to generate the prime implicants using sum-of-products Ternary Decision Diagrams.

When a Boolean function is given in a sum-of-products form, we can generate the sum-of-prime-implicants form of the given function as follows [7]:

Step.1 Generate the sum-of-minterms form of the given function.

Step.2 Generate all implicants of the given function.

Step.3 Remove implicants which subsume other implicants.

Along these steps, we discuss about the generation of the prime implicants with sum-of-products Ternary Decision Diagrams in the following subsections.

6.1.1 Generating Sum-of-Minterms TDDs

When an n-variable Boolean function f is given in a sum-of-products form P, we can generate the sum-of-minterms form Q of f by Shannon Expansion:

$$Q = \{\overline{x_1}, x_1\}\times\{\overline{x_2}, x_2\}\times\cdots\times\{\overline{x_n}, x_n\}\times P$$

as mentioned in section 3.4. For example, when $\overline{x_1}x_2\overline{x_3}+x_1\overline{x_3}x_4+\overline{x_1}x_3+x_1x_3\overline{x_4}+\overline{x_2}\,\overline{x_3}\,\overline{x_4}$ is given as P for f, we can obtain the sum-of-minterms form $\overline{x_1}x_2\overline{x_3}\,\overline{x_4}+\overline{x_1}x_2\overline{x_3}x_4+x_1\overline{x_2}\,\overline{x_3}x_4+x_1x_2\overline{x_3}x_4+\overline{x_1}\,\overline{x_2}x_3\overline{x_4}+\overline{x_1}\,\overline{x_2}x_3x_4+\overline{x_1}x_2x_3\overline{x_4}+\overline{x_1}x_2x_3x_4+x_1\overline{x_2}x_3\overline{x_4}+x_1x_2x_3\overline{x_4}+\overline{x_1}\,\overline{x_2}\,\overline{x_3}\,\overline{x_4}+x_1\overline{x_2}\,\overline{x_3}\,\overline{x_4}$ of the given Boolean function f.

We can easily implement this procedure under Ternary Decision Diagrams, because we can actualize the set operation Cartesian product appeared in the expansion on Ternary Decision Diagrams as we discussed in chapter 3.

6.1.2 Generating All Implicants on TDDs

When an n-variable Boolean function f is given in its sum-of-minterms form Q, we can generate the sum-of-all-implicants form R of f by combining the minterms as shown below:

```
R:=Q
for i:=1 to n begin
  R:=R∪(R/{x̄ᵢ, xᵢ})
  end
```

This procedure is mainly based on the fact that:

$(R/\{\overline{x_i}, x_i\})\ni p$ iff $R\ni\overline{x_i}p$ and $R\ni x_ip$.

It is correspond to the combining of two implicants $\overline{x_i}p$ and x_ip into an implicant p.

As an example of the execution of this procedure, we show the transition of the set R in Figure 6.1 when $\overline{x_1}x_2\overline{x_3}\,\overline{x_4}+\overline{x_1}x_2\overline{x_3}x_4+x_1\overline{x_2}\,\overline{x_3}x_4+x_1x_2\overline{x_3}x_4+\overline{x_1}\,\overline{x_2}x_3\overline{x_4}+\overline{x_1}\,\overline{x_2}x_3x_4+\overline{x_1}x_2x_3\overline{x_4}+\overline{x_1}x_2x_3x_4+x_1\overline{x_2}x_3\overline{x_4}+x_1x_2x_3\overline{x_4}+\overline{x_1}\,\overline{x_2}\,\overline{x_3}\,\overline{x_4}+x_1\overline{x_2}\,\overline{x_3}\,\overline{x_4}$ is given for f as Q. After the exe-

cution, we have obtained the sum-of-all-implicants form $\overline{x_1}x_2\overline{x_3}\,\overline{x_4}+\overline{x_1}x_2\overline{x_3}x_4+x_1\overline{x_2}\,\overline{x_3}x_4+x_1x_2\overline{x_3}x_4+\overline{x_1}\,\overline{x_2}x_3\overline{x_4}+\overline{x_1}\,\overline{x_2}x_3x_4+\overline{x_1}x_2x_3\overline{x_4}+\overline{x_1}x_2x_3x_4+x_1\overline{x_2}x_3\overline{x_4}+x_1x_2x_3\overline{x_4}+\overline{x_1}\,\overline{x_2}\,\overline{x_3}\,\overline{x_4}+x_1\overline{x_2}\,\overline{x_3}\,\overline{x_4}+x_2\overline{x_3}x_4+\overline{x_2}x_3\overline{x_4}+x_2x_3\overline{x_4}+\overline{x_2}\,\overline{x_3}\,\overline{x_4}+\overline{x_1}\,\overline{x_3}\,\overline{x_4}+x_1\overline{x_3}x_4+\overline{x_1}x_3x_4+\overline{x_1}x_3\overline{x_4}+x_1x_3\overline{x_4}+x_3\overline{x_4}+\overline{x_1}\,\overline{x_2}\,\overline{x_4}+\overline{x_1}x_2\overline{x_4}+\overline{x_1}x_2x_4+x_1\overline{x_2}\,\overline{x_4}+\overline{x_1}\,\overline{x_4}+\overline{x_1}\,\overline{x_2}x_3+\overline{x_1}x_2\overline{x_3}+\overline{x_1}x_2x_3+x_1\overline{x_2}\,\overline{x_3}+\overline{x_1}x_3+\overline{x_1}x_2+\overline{x_2}\,\overline{x_4}$ of the given Boolean function f.

We can easily implement this procedure under Ternary Decision Diagrams, because we can actualize the set operations appeared in this procedure, namely union and weak division, on Ternary Decision Diagrams as we discussed in chapter 3.

6.1.3 Removing Redundant Implicants on TDDs

When an n-variable Boolean function f is given in its sum-of-all-implicants form R, we can generate the sum-of-all-prime-implicants form S of f by removing redundant implicants as shown below:

```
S:=R
for i:=1 to n begin
  S:=S−(R/{x̄ᵢ, xᵢ}×{x̄ᵢ, xᵢ})
  end
```

This procedure is mainly based on the fact that:

$(\mathrm{R}/\{\overline{x_i}, x_i\})\ni p$ iff $\mathrm{R}\ni\overline{x_i}p$ and $\mathrm{R}\ni x_ip$.

When $\mathrm{R}\ni\overline{x_i}p$ and $\mathrm{R}\ni x_ip$, R should include an implicant which is subsumed by $\overline{x_i}p$ and x_ip since R consists of all implicants of f. Thus $\overline{x_i}p$ and x_ip can be removed from S.

As an example of the execution of this procedure, we show the transition of the set S in Figure 6.2 when $\overline{x_1}x_2\overline{x_3}\,\overline{x_4}+\overline{x_1}x_2\overline{x_3}x_4+x_1\overline{x_2}\,\overline{x_3}x_4+x_1x_2\overline{x_3}x_4+\overline{x_1}\,\overline{x_2}x_3\overline{x_4}+\overline{x_1}\,\overline{x_2}x_3x_4+\overline{x_1}x_2x_3\overline{x_4}+\overline{x_1}x_2x_3x_4+x_1\overline{x_2}x_3\overline{x_4}+x_1x_2x_3\overline{x_4}+\overline{x_1}\,\overline{x_2}\,\overline{x_3}\,\overline{x_4}+x_1\overline{x_2}\,\overline{x_3}\,\overline{x_4}+x_2\overline{x_3}x_4+\overline{x_2}x_3\overline{x_4}+x_2x_3\overline{x_4}+\overline{x_2}\,\overline{x_3}\,\overline{x_4}+\overline{x_1}\,\overline{x_3}\,\overline{x_4}+x_1\overline{x_3}x_4+\overline{x_1}x_3x_4+\overline{x_1}x_3\overline{x_4}+x_1x_3\overline{x_4}+x_3\overline{x_4}+\overline{x_1}\,\overline{x_2}\,\overline{x_4}+\overline{x_1}x_2\overline{x_4}+\overline{x_1}x_2x_4+x_1\overline{x_2}\,\overline{x_4}+\overline{x_1}\,\overline{x_4}+\overline{x_1}\,\overline{x_2}x_3+\overline{x_1}x_2\overline{x_3}+\overline{x_1}x_2x_3+x_1\overline{x_2}\,\overline{x_3}+\overline{x_1}x_3+\overline{x_1}x_2+\overline{x_2}\,\overline{x_4}$ is given for f as R. After the exe-

cution, we have obtained the sum-of-all-prime-implicants form $x_2\overline{x_3}x_4+x_1\overline{x_3}x_4+x_3\overline{x_4}+\overline{x_1}\,\overline{x_4}+x_1\overline{x_2}\,\overline{x_3}+\overline{x_1}x_3+\overline{x_1}x_2+\overline{x_2}\,\overline{x_4}$ of the given Boolean function f.

We can easily implement this procedure under Ternary Decision Diagrams, because we can actualize the set operations appeared in this procedure, namely difference, weak division, and Cartesian product, on Ternary Decision Diagrams as we discussed in chapter 3.

6.2 Representing Neighfunction on TDDs

Neighfunction [20] is a Boolean function, which is defined as the sum of prime implicants which are subsumed by a particular minterm. The neighfunction is used to optimize sum-of-products forms, especially to find essential and quasi-essential prime implicants in a PLA-optimizer "TACCO" [22]. In this section, we discuss about the method to generate the neighfunction using sum-of-products Ternary Decision Diagrams.

6.2.1 Neighfunction and Its Property

When a Boolean function f and a minterm p which satisfy $f \cdot p = p$ are given, f_p is the **neighfunction** of p in f iff

$$^{\forall}c \text{ s.t. } c \text{ is a prime implicant of } f,\ c \cdot p = p \Leftrightarrow f_p \cdot c = c.$$

For example, when the function f is given as a Karnaugh map in Figure 6.3 and the given minterm p is $\overline{x_1}x_2\overline{x_3}x_4$, the neighfunction f_p is the function described in Figure 6.4.

Here we consider the property of the neighfunction f_p. Let $\dot{p}$ denote the "literal-inverted" minterm of a minterm p (for example, when p is $\overline{x_1}x_2\overline{x_3}x_4$, $\dot{p}$ is $x_1\overline{x_2}x_3\overline{x_4}$). When a minterm q which satisfies $f \cdot q = 0$ is given, the neighfunction f_p has the property that

$f_p \cdot c = 0$, where c is the minimum product which satisfies $c \cdot \dot{p} = \dot{p}$ and $c \cdot q = q$.

For example, when the given Boolean function f is as a Karnaugh map in Figure 6.3 and the given minterms p and q are $\overline{x_1}x_2\overline{x_3}x_4$ and $\overline{x_1}\,\overline{x_2}\,\overline{x_3}x_4$ respectively, the neighfunction f_p should be disjointed from $\overline{x_2}$ as shown in Figure 6.5.

Following to this property, we can easily say that f_p is disjointed from the product of literals which are included only in q but not in p.

6.2.2 Generating Neighfunction on TDDs

When a Boolean function f is given in a sum-of-products form P, and a minterm p which satisfies $f \cdot p = p$ is given in the set of literals $\{\theta_1, \theta_2, \cdots, \theta_n\}$ included in p, we can generate a sum-of-products form Q of the neighfunction f_p as follows:

```
Q:=P̃
for i:=1 to n begin
  Q:=Q∪(Q/{θ_i})
  end
Q:=Q̃
```

As an example of the execution of this procedure, we show the transition of the set Q in Figure 6.6 when $\overline{x_1}x_2\overline{x_3}+x_1\overline{x_3}x_4+\overline{x_1}x_3+x_1x_3\overline{x_4}+\overline{x_2}\,\overline{x_3}\,\overline{x_4}$ is given for f and $\{\overline{x_1}, x_2, \overline{x_3}, x_4\}$ is given for p. After the execution, we have got the sum-of-products form $\overline{x_1}x_2+x_1x_2\overline{x_3}x_4+x_2\overline{x_3}x_4$ which represents the neighfunction shown in Figure 6.4.

We can easily implement this procedure under Ternary Decision Diagrams, because we can actualize the set operations appeared in this procedure, namely complement, union, and weak division, on Ternary Decision Diagrams as we discussed in chapter 3.

6.3 Generating Optimized ROPs on TDDs

In this section, we discuss how we describe optimization techniques for ringsum-of-products forms under the set operations on Ternary Decision Diagrams. Here we consider the

ringsum-of-products Ternary Decision Diagrams, thus available operations are symmetric difference, ringsum product, weak division, and ringsum complement as we discussed in chapter 4.

6.3.1 ROPs Optimization Techniques

There are four categories of the optimization techniques for ringsum-of-products forms, so-called MERGE, RESHAPE, EXPAND, and REDUCE. In ROPs optimizers nowadays, such as presented in [29], we apply these techniques one by one to ringsum-of-products forms, and reduce the number of products in the forms.

i) **MERGE**

$x \oplus \overline{x} \rightarrow 1$, $x \oplus 1 \rightarrow \overline{x}$, $\overline{x} \oplus 1 \rightarrow x$.

ii) **RESHAPE**

$x_i \overline{x_j} \oplus \overline{x_i} \rightarrow \overline{x_i} x_j \oplus \overline{x_j}$, $x_i \oplus \overline{x_j} \rightarrow \overline{x_i} \oplus x_j$, $x_i x_j \oplus \overline{x_i} \leftrightarrow \overline{x_i}\,\overline{x_j} \oplus x_j$, $x_i \oplus x_j \leftrightarrow \overline{x_i} \oplus \overline{x_j}$.

iii) **EXPAND**

$x_i x_j \oplus \overline{x_i}\,\overline{x_j} \rightarrow x_i \oplus x_j$, $\overline{x_i} x_j \oplus x_i \overline{x_j} \rightarrow \overline{x_i} \oplus x_j$, $x_i x_j \oplus \overline{x_i} \rightarrow x_i \overline{x_j} \oplus 1$, $\overline{x_i} x_j \oplus x_i \rightarrow \overline{x_i}\,\overline{x_j} \oplus 1$, $x_i \overline{x_j} \oplus \overline{x_i} \rightarrow x_i x_j \oplus 1$, $\overline{x_i}\,\overline{x_j} \oplus x_i \rightarrow \overline{x_i} x_j \oplus 1$.

iv) **REDUCE**

$x_i \oplus x_j \rightarrow x_i x_j \oplus \overline{x_i}\,\overline{x_j}$, $\overline{x_i} \oplus x_j \rightarrow \overline{x_i} x_j \oplus x_i \overline{x_j}$, $x_i \overline{x_j} \oplus 1 \rightarrow x_i x_j \oplus \overline{x_i}$, $\overline{x_i}\,\overline{x_j} \oplus 1 \rightarrow \overline{x_i} x_j \oplus x_i$, $x_i x_j \oplus 1 \rightarrow x_i \overline{x_j} \oplus \overline{x_i}$, $\overline{x_i} x_j \oplus 1 \rightarrow \overline{x_i}\,\overline{x_j} \oplus x_i$.

Here we note that any of these operations does not change the Boolean function which the ringsum-of-products form expresses, and that the operations in EXPAND have their corresponding inverse operations in REDUCE.

We consider the optimization of $\overline{x_1} x_2 \overline{x_3} x_4 \oplus x_1 \overline{x_3} x_4 \oplus x_1 \overline{x_2}\,\overline{x_3}\,\overline{x_4} \oplus \overline{x_1}\,\overline{x_4} \oplus \overline{x_1} x_3 \oplus x_3 \overline{x_4}$ shown in Figure 6.7 as an example of the application of these techniques for ringsum-of-products forms.

6.3.2 ROPs Optimization Techniques on TDDs

Now we discuss how we realize the optimization techniques in the categories, MERGE, RESHAPE, EXPAND, and REDUCE, by the set operations on ringsum-of-products Ternary Decision Diagrams. In the following, we denote P as a ringsum-of-products form which is required to be optimized.

i) **MERGE**

$x \oplus \overline{x} \rightarrow 1$ is realized as P:=P$\oplus$(P/$\{x, \overline{x}\}$)$\otimes\{x, \overline{x}, 1\}$).

$x \oplus 1 \rightarrow \overline{x}$ is realized as P:=P$\oplus$(P/$\{x, 1\}\otimes\{x, \overline{x}, 1\}$).

$\overline{x} \oplus 1 \rightarrow x$ is realized as P:=P$\oplus$(P/$\{\overline{x}, 1\}\otimes\{x, \overline{x}, 1\}$).

ii) **RESHAPE**

$x_i\overline{x_j} \oplus \overline{x_i} \rightarrow \overline{x_i}x_j \oplus \overline{x_j}$ is realized as P:=P$\oplus$(P/$\{x_i\overline{x_j}, \overline{x_i}\}\otimes\{x_i\overline{x_j}, \overline{x_i}, \overline{x_i}x_j, \overline{x_j}\}$).

$x_i \oplus \overline{x_j} \rightarrow \overline{x_i} \oplus x_j$ is realized as P:=P$\oplus$(P/$\{x_i, \overline{x_j}\}\otimes\{x_i, \overline{x_j}, \overline{x_i}, x_j\}$).

$x_ix_j \oplus \overline{x_i} \rightarrow \overline{x_i}\,\overline{x_j} \oplus x_j$ is realized as P:=P$\oplus$(P/$\{x_ix_j, \overline{x_i}\}\otimes\{x_ix_j, \overline{x_i}, \overline{x_i}\,\overline{x_j}, x_j\}$).

$\overline{x_i}\,\overline{x_j} \oplus x_j \rightarrow x_ix_j \oplus \overline{x_i}$ is realized as P:=P$\oplus$(P/$\{\overline{x_i}\,\overline{x_j}, x_j\}\otimes\{x_ix_j, \overline{x_i}, \overline{x_i}\,\overline{x_j}, x_j\}$).

$x_i \oplus x_j \rightarrow \overline{x_i} \oplus \overline{x_j}$ is realized as P:=P$\oplus$(P/$\{x_i, x_j\}\otimes\{x_i, x_j, \overline{x_i}, \overline{x_j}\}$).

$\overline{x_i} \oplus \overline{x_j} \rightarrow x_i \oplus x_j$ is realized as P:=P$\oplus$(P/$\{\overline{x_i}, \overline{x_j}\}\otimes\{x_i, x_j, \overline{x_i}, \overline{x_j}\}$).

iii) **EXPAND**

$x_ix_j \oplus \overline{x_i}\,\overline{x_j} \rightarrow x_i \oplus x_j$ is realized as P:=P$\oplus$(P/$\{x_ix_j, \overline{x_i}\,\overline{x_j}\}\otimes\{x_ix_j, \overline{x_i}\,\overline{x_j}, x_i, x_j\}$).

$\overline{x_i}x_j \oplus x_i\overline{x_j} \rightarrow \overline{x_i} \oplus x_j$ is realized as P:=P$\oplus$(P/$\{\overline{x_i}x_j, x_i\overline{x_j}\}\otimes\{\overline{x_i}x_j, x_i\overline{x_j}, \overline{x_i}, x_j\}$).

$x_ix_j \oplus \overline{x_i} \rightarrow x_i\overline{x_j} \oplus 1$ is realized as P:=P$\oplus$(P/$\{x_ix_j, \overline{x_i}\}\otimes\{x_ix_j, \overline{x_i}, x_i\overline{x_j}, 1\}$).

$\overline{x_i}x_j \oplus x_i \rightarrow \overline{x_i}\,\overline{x_j} \oplus 1$ is realized as P:=P$\oplus$(P/$\{\overline{x_i}x_j, x_i\}\otimes\{\overline{x_i}x_j, x_i, \overline{x_i}\,\overline{x_j}, 1\}$).

$x_i\overline{x_j} \oplus \overline{x_i} \rightarrow x_ix_j \oplus 1$ is realized as P:=P$\oplus$(P/$\{x_i\overline{x_j}, \overline{x_i}\}\otimes\{x_i\overline{x_j}, \overline{x_i}, x_ix_j, 1\}$).

$\overline{x_i}\,\overline{x_j} \oplus x_i \rightarrow \overline{x_i}x_j \oplus 1$ is realized as P:=P$\oplus$(P/$\{\overline{x_i}\,\overline{x_j}, x_i\}\otimes\{\overline{x_i}\,\overline{x_j}, x_i, \overline{x_i}x_j, 1\}$).

iv) **REDUCE**

$x_i \oplus x_j \rightarrow x_ix_j \oplus \overline{x_i}\,\overline{x_j}$ is realized as P:=P$\oplus$(P/$\{x_i, x_j\}\otimes\{x_i, x_j, x_ix_j, \overline{x_i}\,\overline{x_j}\}$).

$\overline{x_i} \oplus x_j \rightarrow \overline{x_i}x_j \oplus x_i\overline{x_j}$ is realized as P:=P$\oplus$(P/$\{\overline{x_i}, x_j\}\otimes\{\overline{x_i}, x_j, \overline{x_i}x_j, x_i\overline{x_j}\}$).

$x_i\overline{x_j}\oplus 1 \rightarrow x_ix_j\oplus\overline{x_i}$ is realized as P:=P$\oplus$(P/$\{x_i\overline{x_j}, 1\}\otimes\{x_i\overline{x_j}, 1, x_ix_j, \overline{x_i}\}$).

$\overline{x_i}\,\overline{x_j}\oplus 1 \rightarrow \overline{x_i}x_j\oplus x_i$ is realized as P:=P$\oplus$(P/$\{\overline{x_i}\,\overline{x_j}, 1\}\otimes\{\overline{x_i}\,\overline{x_j}, 1, \overline{x_i}x_j, x_i\}$).

$x_ix_j\oplus 1 \rightarrow x_i\overline{x_j}\oplus\overline{x_i}$ is realized as P:=P$\oplus$(P/$\{x_ix_j, 1\}\otimes\{x_ix_j, 1, x_i\overline{x_j}, \overline{x_i}\}$).

$\overline{x_i}x_j\oplus 1 \rightarrow \overline{x_i}\,\overline{x_j}\oplus x_i$ is realized as P:=P$\oplus$(P/$\{\overline{x_i}x_j, 1\}\otimes\{\overline{x_i}x_j, 1, \overline{x_i}\,\overline{x_j}, x_i\}$).

According to this realization, Ternary Decision Diagrams can actualize any of the ROPs optimization techniques by the set operations, which are symmetric difference, weak division, and ringsum product. Figure 6.8 shows an example to optimize $\overline{x_1}x_2\overline{x_3}x_4\oplus x_1\overline{x_3}x_4 \oplus x_1\overline{x_2}\,\overline{x_3}\,\overline{x_4}\oplus\overline{x_1}\,\overline{x_4}\oplus\overline{x_1}x_3\oplus x_3\overline{x_4}$ by the set operations under Ternary Decision Diagrams. In this example, the number of products is reduced from 6 to 4 with 18 set operations.

$$\Downarrow \mathrm{R}:=\mathrm{Q}$$

$$\{\overline{x_1}x_2\overline{x_3}\,\overline{x_4},\ \overline{x_1}x_2\overline{x_3}x_4,\ x_1\overline{x_2}\,\overline{x_3}x_4,\ x_1x_2\overline{x_3}x_4,\ \overline{x_1}\,\overline{x_2}x_3\overline{x_4},\ \overline{x_1}\,\overline{x_2}x_3x_4,\ \overline{x_1}x_2x_3\overline{x_4},\ \overline{x_1}x_2x_3x_4,$$
$$x_1\overline{x_2}x_3\overline{x_4},\ x_1x_2x_3\overline{x_4},\ \overline{x_1}\,\overline{x_2}\,\overline{x_3}\,\overline{x_4},\ x_1\overline{x_2}\,\overline{x_3}\,\overline{x_4}\}$$

$$\Downarrow \mathrm{R}:=\mathrm{R}\cup(\mathrm{R}/\{\overline{x_1},\ x_1\})$$

$$\{\overline{x_1}x_2\overline{x_3}\,\overline{x_4},\ \overline{x_1}x_2\overline{x_3}x_4,\ x_1\overline{x_2}\,\overline{x_3}x_4,\ x_1x_2\overline{x_3}x_4,\ \overline{x_1}\,\overline{x_2}x_3\overline{x_4},\ \overline{x_1}\,\overline{x_2}x_3x_4,\ \overline{x_1}x_2x_3\overline{x_4},\ \overline{x_1}x_2x_3x_4,$$
$$x_1\overline{x_2}x_3\overline{x_4},\ x_1x_2x_3\overline{x_4},\ \overline{x_1}\,\overline{x_2}\,\overline{x_3}\,\overline{x_4},\ x_1\overline{x_2}\,\overline{x_3}\,\overline{x_4},\ x_2\overline{x_3}x_4,\ \overline{x_2}x_3\overline{x_4},\ x_2x_3\overline{x_4},\ \overline{x_2}\,\overline{x_3}\,\overline{x_4}\}$$

$$\Downarrow \mathrm{R}:=\mathrm{R}\cup(\mathrm{R}/\{\overline{x_2},\ x_2\})$$

$$\{\overline{x_1}x_2\overline{x_3}\,\overline{x_4},\ \overline{x_1}x_2\overline{x_3}x_4,\ x_1\overline{x_2}\,\overline{x_3}x_4,\ x_1x_2\overline{x_3}x_4,\ \overline{x_1}\,\overline{x_2}x_3\overline{x_4},\ \overline{x_1}\,\overline{x_2}x_3x_4,\ \overline{x_1}x_2x_3\overline{x_4},\ \overline{x_1}x_2x_3x_4,$$
$$x_1\overline{x_2}x_3\overline{x_4},\ x_1x_2x_3\overline{x_4},\ \overline{x_1}\,\overline{x_2}\,\overline{x_3}\,\overline{x_4},\ x_1\overline{x_2}\,\overline{x_3}\,\overline{x_4},\ x_2\overline{x_3}x_4,\ \overline{x_2}x_3\overline{x_4},\ x_2x_3\overline{x_4},\ \overline{x_2}\,\overline{x_3}\,\overline{x_4},\ \overline{x_1}\,\overline{x_3}\,\overline{x_4},$$
$$x_1\overline{x_3}x_4,\ \overline{x_1}x_3x_4,\ \overline{x_1}x_3\overline{x_4},\ x_1x_3\overline{x_4},\ x_3\overline{x_4}\}$$

$$\Downarrow \mathrm{R}:=\mathrm{R}\cup(\mathrm{R}/\{\overline{x_3},\ x_3\})$$

$$\{\overline{x_1}x_2\overline{x_3}\,\overline{x_4},\ \overline{x_1}x_2\overline{x_3}x_4,\ x_1\overline{x_2}\,\overline{x_3}x_4,\ x_1x_2\overline{x_3}x_4,\ \overline{x_1}\,\overline{x_2}x_3\overline{x_4},\ \overline{x_1}\,\overline{x_2}x_3x_4,\ \overline{x_1}x_2x_3\overline{x_4},\ \overline{x_1}x_2x_3x_4,$$
$$x_1\overline{x_2}x_3\overline{x_4},\ x_1x_2x_3\overline{x_4},\ \overline{x_1}\,\overline{x_2}\,\overline{x_3}\,\overline{x_4},\ x_1\overline{x_2}\,\overline{x_3}\,\overline{x_4},\ x_2\overline{x_3}x_4,\ \overline{x_2}x_3\overline{x_4},\ x_2x_3\overline{x_4},\ \overline{x_2}\,\overline{x_3}\,\overline{x_4},\ \overline{x_1}\,\overline{x_3}\,\overline{x_4},$$
$$x_1\overline{x_3}x_4,\ \overline{x_1}x_3x_4,\ \overline{x_1}x_3\overline{x_4},\ x_1x_3\overline{x_4},\ x_3\overline{x_4},\ \overline{x_1}\,\overline{x_2}\,\overline{x_4},\ \overline{x_1}x_2\overline{x_4},\ \overline{x_1}x_2x_4,\ x_1\overline{x_2}\,\overline{x_4},\ \overline{x_1}\,\overline{x_4},\ \overline{x_2}\,\overline{x_4}\}$$

$$\Downarrow \mathrm{R}:=\mathrm{R}\cup(\mathrm{R}/\{\overline{x_4},\ x_4\})$$

$$\{\overline{x_1}x_2\overline{x_3}\,\overline{x_4},\ \overline{x_1}x_2\overline{x_3}x_4,\ x_1\overline{x_2}\,\overline{x_3}x_4,\ x_1x_2\overline{x_3}x_4,\ \overline{x_1}\,\overline{x_2}x_3\overline{x_4},\ \overline{x_1}\,\overline{x_2}x_3x_4,\ \overline{x_1}x_2x_3\overline{x_4},\ \overline{x_1}x_2x_3x_4,$$
$$x_1\overline{x_2}x_3\overline{x_4},\ x_1x_2x_3\overline{x_4},\ \overline{x_1}\,\overline{x_2}\,\overline{x_3}\,\overline{x_4},\ x_1\overline{x_2}\,\overline{x_3}\,\overline{x_4},\ x_2\overline{x_3}x_4,\ \overline{x_2}x_3\overline{x_4},\ x_2x_3\overline{x_4},\ \overline{x_2}\,\overline{x_3}\,\overline{x_4},\ \overline{x_1}\,\overline{x_3}\,\overline{x_4},$$
$$x_1\overline{x_3}x_4,\ \overline{x_1}x_3x_4,\ \overline{x_1}x_3\overline{x_4},\ x_1x_3\overline{x_4},\ x_3\overline{x_4},\ \overline{x_1}\,\overline{x_2}\,\overline{x_4},\ \overline{x_1}x_2\overline{x_4},\ \overline{x_1}x_2x_4,\ x_1\overline{x_2}\,\overline{x_4},\ \overline{x_1}\,\overline{x_4},\ \overline{x_1}\,\overline{x_2}x_3,$$
$$\overline{x_1}x_2\overline{x_3},\ \overline{x_1}x_2x_3,\ x_1\overline{x_2}\,\overline{x_3},\ \overline{x_1}x_3,\ \overline{x_1}x_2,\ \overline{x_2}\,\overline{x_4}\}$$

Figure 6.1: Generating all implicants

$$\Downarrow \text{S:=R}$$

$$\{\overline{x_1}x_2\overline{x_3}\,\overline{x_4},\ \overline{x_1}x_2\overline{x_3}x_4,\ x_1\overline{x_2}\,\overline{x_3}x_4,\ x_1x_2\overline{x_3}x_4,\ \overline{x_1}\,\overline{x_2}x_3\overline{x_4},\ \overline{x_1}\,\overline{x_2}x_3x_4,\ \overline{x_1}x_2x_3\overline{x_4},\ \overline{x_1}x_2x_3x_4,$$
$$x_1\overline{x_2}x_3\overline{x_4},\ x_1x_2x_3\overline{x_4},\ \overline{x_1}\,\overline{x_2}\,\overline{x_3}\,\overline{x_4},\ x_1\overline{x_2}\,\overline{x_3}\,\overline{x_4},\ x_2\overline{x_3}x_4,\ \overline{x_2}x_3\overline{x_4},\ x_2x_3\overline{x_4},\ \overline{x_2}\,\overline{x_3}\,\overline{x_4},\ \overline{x_1}\,\overline{x_3}\,\overline{x_4},$$
$$x_1\overline{x_3}x_4,\ \overline{x_1}x_3x_4,\ \overline{x_1}x_3\overline{x_4},\ x_1x_3\overline{x_4},\ x_3\overline{x_4},\ \overline{x_1}\,\overline{x_2}\,\overline{x_4},\ \overline{x_1}x_2\overline{x_4},\ \overline{x_1}x_2x_4,\ x_1\overline{x_2}\,\overline{x_4},\ \overline{x_1}\,\overline{x_4},\ \overline{x_1}\,\overline{x_2}x_3,$$
$$\overline{x_1}x_2\overline{x_3},\ \overline{x_1}x_2x_3,\ x_1\overline{x_2}\,\overline{x_3},\ \overline{x_1}x_3,\ \overline{x_1}x_2,\ \overline{x_2}\,\overline{x_4}\}$$

$$\Downarrow \text{S:=S}-(\text{R}/\{\overline{x_1},\ x_1\}\times\{\overline{x_1},\ x_1\})$$

$$\{\overline{x_1}x_2\overline{x_3}\,\overline{x_4},\ x_1\overline{x_2}\,\overline{x_3}x_4,\ x_1x_2\overline{x_3}x_4,\ \overline{x_1}\,\overline{x_2}x_3\overline{x_4},\ x_2\overline{x_3}x_4,\ \overline{x_2}x_3\overline{x_4},\ x_2x_3\overline{x_4},\ \overline{x_2}\,\overline{x_3}\,\overline{x_4},\ \overline{x_1}\,\overline{x_3}\,\overline{x_4},$$
$$x_1\overline{x_3}x_4,\ \overline{x_1}x_3x_4,\ x_3\overline{x_4},\ \overline{x_1}x_2\overline{x_4},\ \overline{x_1}x_2x_4,\ \overline{x_1}\,\overline{x_4},\ \overline{x_1}\,\overline{x_2}x_3,\ \overline{x_1}x_2\overline{x_3},\ \overline{x_1}x_2x_3,\ x_1\overline{x_2}\,\overline{x_3},\ \overline{x_1}x_3,\ \overline{x_1}x_2,$$
$$\overline{x_2}\,\overline{x_4}\}$$

$$\Downarrow \text{S:=S}-(\text{R}/\{\overline{x_2},\ x_2\}\times\{\overline{x_2},\ x_2\})$$

$$\{x_2\overline{x_3}x_4,\ \overline{x_2}\,\overline{x_3}\,\overline{x_4},\ \overline{x_1}\,\overline{x_3}\,\overline{x_4},\ x_1\overline{x_3}x_4,\ \overline{x_1}x_3x_4,\ x_3\overline{x_4},\ \overline{x_1}x_2x_4,\ \overline{x_1}\,\overline{x_4},\ \overline{x_1}x_2\overline{x_3},\ x_1\overline{x_2}\,\overline{x_3},\ \overline{x_1}x_3,$$
$$\overline{x_1}x_2,\ \overline{x_2}\,\overline{x_4}\}$$

$$\Downarrow \text{S:=S}-(\text{R}/\{\overline{x_3},\ x_3\}\times\{\overline{x_3},\ x_3\})$$

$$\{x_2\overline{x_3}x_4,\ x_1\overline{x_3}x_4,\ \overline{x_1}x_3x_4,\ x_3\overline{x_4},\ \overline{x_1}x_2x_4,\ \overline{x_1}\,\overline{x_4},\ x_1\overline{x_2}\,\overline{x_3},\ \overline{x_1}x_3,\ \overline{x_1}x_2,\ \overline{x_2}\,\overline{x_4}\}$$

$$\Downarrow \text{S:=S}-(\text{R}/\{\overline{x_4},\ x_4\}\times\{\overline{x_4},\ x_4\})$$

$$\{x_2\overline{x_3}x_4,\ x_1\overline{x_3}x_4,\ x_3\overline{x_4},\ \overline{x_1}\,\overline{x_4},\ x_1\overline{x_2}\,\overline{x_3},\ \overline{x_1}x_3,\ \overline{x_1}x_2,\ \overline{x_2}\,\overline{x_4}\}$$

Figure 6.2: Removing redundant implicants

x_1x_2 \ x_3x_4	00	01	11	10
00	1	0	1	1
01	1	1	1	1
11	0	1	0	1
10	1	1	0	1

Figure 6.3: Given Boolean function f

x_1x_2 \ x_3x_4	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	0	1	0	0
10	0	0	0	0

Figure 6.4: Neighfunction $f_{\overline{x_1}x_2\overline{x_3}x_4}$

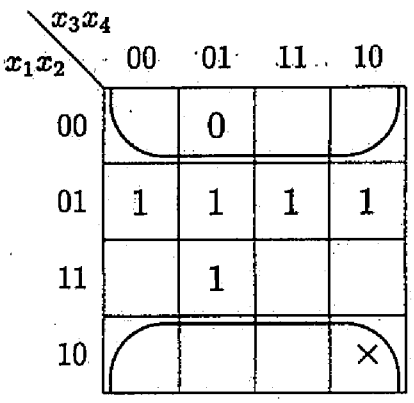


Figure 6.5: Property of Neighfunction

$$\Downarrow \mathrm{Q}:=\tilde{\mathrm{P}}$$

$$\{\overline{x_1}\,\overline{x_2}\,\overline{x_3}x_4,\ x_1x_3x_4,\ x_1\overline{x_2}x_3x_4,\ x_1x_2x_3x_4,\ x_1x_2\overline{x_3}\,\overline{x_4}\}$$

$$\Downarrow \mathrm{Q}:=\mathrm{Q}\cup(\mathrm{Q}/\{\overline{x_1}\})$$

$$\{\overline{x_1}\,\overline{x_2}\,\overline{x_3}x_4,\ \overline{x_2}\,\overline{x_3}x_4,\ x_1x_3x_4,\ x_1\overline{x_2}x_3x_4,\ x_1x_2x_3x_4,\ x_1x_2\overline{x_3}\,\overline{x_4}\}$$

$$\Downarrow \mathrm{Q}:=\mathrm{Q}\cup(\mathrm{Q}/\{x_2\})$$

$$\{\overline{x_1}\,\overline{x_2}\,\overline{x_3}x_4,\ \overline{x_2}\,\overline{x_3}x_4,\ x_1x_3x_4,\ x_1\overline{x_2}x_3x_4,\ x_1x_2x_3x_4,\ x_1x_2\overline{x_3}\,\overline{x_4},\ x_1\overline{x_3}\,\overline{x_4}\}$$

$$\Downarrow \mathrm{Q}:=\mathrm{Q}\cup(\mathrm{Q}/\{\overline{x_3}\})$$

$$\{\overline{x_1}\,\overline{x_2}\,\overline{x_3}x_4,\ \overline{x_1}\,\overline{x_2}x_4,\ \overline{x_2}\,\overline{x_3}x_4,\ \overline{x_2}x_4,\ x_1x_3x_4,\ x_1\overline{x_2}x_3x_4,\ x_1x_2x_3x_4,\ x_1x_2\overline{x_3}\,\overline{x_4},\ x_1x_2\overline{x_4},$$
$$x_1\overline{x_3}\,\overline{x_4},\ x_1\overline{x_4}\}$$

$$\Downarrow \mathrm{Q}:=\mathrm{Q}\cup(\mathrm{Q}/\{x_4\})$$

$$\{\overline{x_1}\,\overline{x_2}\,\overline{x_3}x_4,\ \overline{x_1}\,\overline{x_2}\,\overline{x_3},\ \overline{x_1}\,\overline{x_2}x_4,\ \overline{x_1}\,\overline{x_2},\ \overline{x_2}\,\overline{x_3}x_4,\ \overline{x_2}\,\overline{x_3},\ \overline{x_2}x_4,\ \overline{x_2},\ x_1x_3x_4,\ x_1x_3,\ x_1\overline{x_2}x_3x_4,$$
$$x_1\overline{x_2}x_3,\ x_1x_2x_3x_4,\ x_1x_2x_3,\ x_1x_2\overline{x_3}\,\overline{x_4},\ x_1x_2\overline{x_4},\ x_1\overline{x_3}\,\overline{x_4},\ x_1\overline{x_4}\}$$

$$\Downarrow \mathrm{Q}:=\tilde{\mathrm{Q}}$$

$$\{\overline{x_1}x_2,\ x_1x_2\overline{x_3}x_4,\ x_2\overline{x_3}x_4\}$$

Figure 6.6: Generating neighfunction

$$\overline{x_1}x_2\overline{x_3}x_4 \oplus x_1\overline{x_3}x_4 \oplus x_1\overline{x_2}\,\overline{x_3}\,\overline{x_4} \oplus \overline{x_1}\,\overline{x_4} \oplus \overline{x_1}x_3 \oplus x_3\overline{x_4}$$

$$\Downarrow \text{EXPAND } (\overline{x_1}x_2\overline{x_3}x_4 \oplus x_1\overline{x_3}x_4 \rightarrow \overline{x_1}\,\overline{x_2}\,\overline{x_3}x_4 \oplus \overline{x_3}x_4)$$

$$\overline{x_1}\,\overline{x_2}\,\overline{x_3}x_4 \oplus \overline{x_3}x_4 \oplus x_1\overline{x_2}\,\overline{x_3}\,\overline{x_4} \oplus \overline{x_1}\,\overline{x_4} \oplus \overline{x_1}x_3 \oplus x_3\overline{x_4}$$

$$\Downarrow \text{RESHAPE } (\overline{x_1}\,\overline{x_4} \oplus x_3\overline{x_4} \rightarrow x_1\overline{x_4} \oplus \overline{x_3}\,\overline{x_4})$$

$$\overline{x_1}\,\overline{x_2}\,\overline{x_3}x_4 \oplus \overline{x_3}x_4 \oplus x_1\overline{x_2}\,\overline{x_3}\,\overline{x_4} \oplus x_1\overline{x_4} \oplus \overline{x_3}\,\overline{x_4} \oplus \overline{x_1}x_3$$

$$\Downarrow \text{MERGE } (\overline{x_3}x_4 \oplus \overline{x_3}\,\overline{x_4} \rightarrow \overline{x_3})$$

$$\overline{x_1}\,\overline{x_2}\,\overline{x_3}x_4 \oplus x_1\overline{x_2}\,\overline{x_3}\,\overline{x_4} \oplus x_1\overline{x_4} \oplus \overline{x_1}x_3 \oplus \overline{x_3}$$

$$\Downarrow \text{REDUCE } (x_1\overline{x_2}\,\overline{x_3}\,\overline{x_4} \oplus x_1\overline{x_4} \rightarrow x_1x_2\overline{x_3}\,\overline{x_4} \oplus x_1x_3\overline{x_4})$$

$$\overline{x_1}\,\overline{x_2}\,\overline{x_3}x_4 \oplus \overline{x_1}x_3 \oplus \overline{x_3} \oplus x_1x_2\overline{x_3}\,\overline{x_4} \oplus x_1x_3\overline{x_4}$$

$$\Downarrow \text{EXPAND } (x_1x_3\overline{x_4} \oplus \overline{x_1}x_3 \rightarrow x_1x_3x_4 \oplus x_3)$$

$$\overline{x_1}\,\overline{x_2}\,\overline{x_3}x_4 \oplus \overline{x_3} \oplus x_1x_2\overline{x_3}\,\overline{x_4} \oplus x_1x_3x_4 \oplus x_3$$

$$\Downarrow \text{MERGE } (x_3 \oplus \overline{x_3} \rightarrow 1)$$

$$\overline{x_1}\,\overline{x_2}\,\overline{x_3}x_4 \oplus x_1x_2\overline{x_3}\,\overline{x_4} \oplus x_1x_3x_4 \oplus 1$$

Figure 6.7: Optimization of ROP

$$\{\overline{x_1}x_2\overline{x_3}x_4,\ x_1\overline{x_3}x_4,\ x_1\overline{x_2}\,\overline{x_3}\,\overline{x_4},\ \overline{x_1}\,\overline{x_4},\ \overline{x_1}x_3,\ x_3\overline{x_4}\}$$

$$\Downarrow \mathrm{P}:=\mathrm{P}\oplus(\mathrm{P}/\{\overline{x_1}x_2,\ x_1\}\otimes\{\overline{x_1}x_2,\ x_1\ \overline{x_1}\,\overline{x_2},\ 1\})$$

$$\{\overline{x_1}\,\overline{x_2}\,\overline{x_3}x_4,\ \overline{x_3}x_4,\ x_1\overline{x_2}\,\overline{x_3}\,\overline{x_4},\ \overline{x_1}\,\overline{x_4},\ \overline{x_1}x_3,\ x_3\overline{x_4}\}$$

$$\Downarrow \mathrm{P}:=\mathrm{P}\oplus(\mathrm{P}/\{\overline{x_1},\ x_3\}\otimes\{\overline{x_1},\ x_3,\ x_1,\ \overline{x_3}\})$$

$$\{\overline{x_1}\,\overline{x_2}\,\overline{x_3}x_4,\ \overline{x_3}x_4,\ x_1\overline{x_2}\,\overline{x_3}\,\overline{x_4},\ x_1\overline{x_4},\ \overline{x_3}\,\overline{x_4},\ \overline{x_1}x_3\}$$

$$\Downarrow \mathrm{P}:=\mathrm{P}\oplus(\mathrm{P}/\{x_4,\ \overline{x_4}\}\otimes\{x_4,\ \overline{x_4},\ 1\})$$

$$\{\overline{x_1}\,\overline{x_2}\,\overline{x_3}x_4,\ x_1\overline{x_2}\,\overline{x_3}\,\overline{x_4},\ x_1\overline{x_4},\ \overline{x_1}x_3,\ \overline{x_3}\}$$

$$\Downarrow \mathrm{P}:=\mathrm{P}\oplus(\mathrm{P}/\{\overline{x_2}\,\overline{x_3},\ 1\}\otimes\{\overline{x_2}\,\overline{x_3},\ 1,\ x_2\overline{x_3},\ x_3\})$$

$$\{\overline{x_1}\,\overline{x_2}\,\overline{x_3}x_4,\ \overline{x_1}x_3,\ \overline{x_3},\ x_1x_2\overline{x_3}\,\overline{x_4},\ x_1x_3\overline{x_4}\}$$

$$\Downarrow \mathrm{P}:=\mathrm{P}\oplus(\mathrm{P}/\{x_1\overline{x_4},\ \overline{x_1}\}\otimes\{x_1\overline{x_4},\ \overline{x_1},\ x_1x_4,\ 1\})$$

$$\{\overline{x_1}\,\overline{x_2}\,\overline{x_3}x_4,\ \overline{x_3},\ x_1x_2\overline{x_3}\,\overline{x_4},\ x_1x_3x_4,\ x_3\}$$

$$\Downarrow \mathrm{P}:=\mathrm{P}\oplus(\mathrm{P}/\{x_3,\ \overline{x_3}\}\otimes\{x_3,\ \overline{x_3},\ 1\})$$

$$\{\overline{x_1}\,\overline{x_2}\,\overline{x_3}x_4,\ x_1x_2\overline{x_3}\,\overline{x_4},\ x_1x_3x_4,\ 1\}$$

Figure 6.8: Optimization of ROP on TDDs

Chapter 7

Conclusion

In this thesis we have discussed theoretical aspects of Ternary Decision Diagrams, their implementations under UNIX C, and their applications on logic synthesis.

In chapter 2, we have defined Ternary Decision Diagrams from the mathematical point of view. We have discussed the three rules, such as Variable Ordering Rule, Node Reduction Rule, and Node Unification Rule, to make Ternary Decision Diagrams canonical for sets of products. We have mentioned about the brief definitions of Binary Decision Diagrams and Quasi-reduced Binary Decision Diagrams in the chapter.

In chapters 3 and 4, we have discussed about Ternary Decision Diagrams, where we regard TDDs representing sum-of-products forms and ringsum-of-products forms, respectively. We have defined 9 recursive procedures to actualize the essential set operations on sum-of-products forms and ringsum-of-products forms, such as the procedure for union, the procedure for symmetric difference, the procedure for Cartesian product, the procedure for ringsum product, the procedure for weak division, the procedure for intersection, the procedure for difference, the procedure for complement, and the procedure for ringsum complement. We have discussed the manipulations of combinatorial circuits on sum-of-products TDDs and ringsum-of-products TDDs. `AND`-, `OR`-, `XOR`- and `NOT`-gates in the circuits are directly manipulated by the procedures for the essential set operations. We

have also discussed how we convert Ternary Decision Diagrams to Binary Decision Diagrams, vice versa. We emphasize here that the conversions can be realized within the procedures for Ternary Decision Diagrams.

In chapter 5, we have discussed about the implementation techniques of the Ternary Decision Diagrams Library under UNIX C. 22 routines and two header files compose the Ternary Decision Diagrams Library, and they are the fruit of many effective techniques. The experimental result shown in this chapter indicates the size of Ternary Decision Diagrams is smaller than the size of Binary Decision Diagrams on the average, so Ternary Decision Diagrams save memory usage to store Boolean functions.

In chapter 6, we have discussed how we use Ternary Decision Diagrams for three applications in the field of logic synthesis. First, we have discussed about the method to generate all prime implicants of a given Boolean function on sum-of-products Ternary Decision Diagrams. We have materialized the procedure to generate prime implicants in three blocks, namely, the block to generate sum-of-minterms Ternary Decision Diagrams, the block to generate all implicants on Ternary Decision Diagrams, and the block to remove redundant implicants on Ternary Decision Diagrams. Second, we have discussed about the method to generate neighfunctions on sum-of-products Ternary Decision Diagrams. We have actualized the procedure to generate neighfunctions using the set operations implemented on Ternary Decision Diagrams, including the operations of weak division, union, and complement. Third, we have discussed how we describe optimization techniques for ringsum-of-products forms, such as MERGE, RESHAPE, EXPAND, and REDUCE, under the set operations on Ternary Decision Diagrams.

Throughout this thesis, we have realized that Ternary Decision Diagrams have enormous activity on the field of VLSI design. We have got to know the effectiveness of weak division, which is already implemented on Ternary Decision Diagrams, through this discussion to apply Ternary Decision Diagrams for the problems on logic synthesis. The implemented Ternary Decision Diagram Library is very portable and convenient to used in CAD systems. The research on Ternary Decision Diagrams will contribute the devel-

opment of the researches into logic synthesis.

Bibliography

[1] A. Nakajima and M. Hanzawa, "The Theory of Equivalent Transformation of Simple Partial Paths in the Relay Circuit (Part I)," *Journal of the Institute of Telegraph and Telephone Engineers of Japan*, no. 165, pp. 1087–1093, 1936 (in Japanese).

[2] C.E. Shannon, "A Symbolic Analysis of Relay and Switching Circuits," *AIEE Transactions*, vol. 57, pp. 713–723, 1938.

[3] W.V. Quine, "The Problem of Simplifying Truth Functions," *The American Mathematical Monthly*, vol. 59, pp. 521–531, 1952.

[4] S. Reed, "A Class of Multiple-Error-Correcting Codes and the Decoding Scheme," *IRE Transactions of the 1954 Symposium on Information Theory*, pp. 38–49, 1954.

[5] D.E. Muller, "Application of Boolean Algebra to Switching Circuit Design and to Error Detection," *IRE Transactions on Electronic Computers*, vol. EC-3, pp. 6–12, 1954.

[6] W.V. Quine, "A Way to Simplify Truth Functions," *The American Mathematical Monthly*, vol. 62, pp. 627–631, 1955.

[7] E.J. McCluskey, Jr., "Minimization of Boolean Functions," *The Bell System Technical Journal*, vol. 35, pp. 1417–1444, 1956.

[8] A. Mukhopadhyay and G. Schmitz, "Minimization of Exclusive Or and Logical Equivalence Switching Circuits," *IEEE Transactions on Computers*, vol. C-19, pp. 132–140, 1970.

[9] H. Fleisher, A. Weinberger, and V. Winkler, "The Writable Personalized Chip," *Computer Design*, vol. 9, pp. 59-66, 1970.

[10] S.J. Hong, R.G. Cain, and D.L. Ostapko, "MINI: A Heuristic Approach for Logic Minimization," *IBM Journal of Research and Development*, vol. 18, pp. 443–458, 1974.

[11] S.B. Akers, "Binary Decision Diagrams," *IEEE Transactions on Computers*, vol. C-27, pp. 509–516, 1978.

[12] G. Papakonstantinou, "Minimization of Modulo-2 Sum of Product," *IEEE Transactions on Computers*, vol. C-28, pp. 163–167, 1979.

[13] R.K. Brayton, G.D. Hatchel, C.T. McMullen, and A. Sangiovanni-Vincentelli, "ESPRESSO-II: A New Logic Minimizer of Programmable Logic Arrays," *Proceedings of Custom Integrated Circuits Conference*, pp. 370–376, 1984.

[14] N.N. Biswas and B. Gurunath, "BANGALORE: An Algorithm for Optimal Minimization of Programmable Logic Arrays," *International Journal of Electronics*, pp. 709–725, 1986.

[15] M.R. Dagenais, V.K. Agarwal, and N.C. Rumin, "McBOOLE: A New Procedure for Exact Logic Minimization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. CAD-5, pp. 229–238, 1986.

[16] R.E. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation," *IEEE Transactions on Computers*, vol. C-35, pp. 677–691, 1986.

[17] A. Tran, "Graphical Method for the Conversion of Minterms to Reed-Muller Coefficients and the Minimisation of Exclusive-Or Switching Functions," *IEE Proceedings Part E*, vol. 134, pp. 93–99, 1987.

[18] S. Malik, A.R. Wang, R.K. Brayton, and A. Sangiovanni-Vincentelli, "Logic Verification Using Binary Decision Diagrams in a Logic Synthesis Environment," in *Proceedings of ICCAD'88*, pp. 6–9, 1988.

[19] Y. Matsunaga, M. Fujita, "Multi-level Logic Optimization Using Binary Decision Diagrams," in *Proceedings of ICCAD'89*, pp. 556–559, 1989.

[20] K. Yasuoka, N. Takagi, and S. Yajima, "Logic Function Minimization Using Neighfunctions," *Record of the 1990 IEICE Spring Conference*, pp. A-104, 1990.

[21] T. Sasao, "EXMIN: A Simplification Algorithm for Exclusive-OR-Sum-of-Products Expressions for Multiple-Valued Input Two-Valued Output Functions," *Proceedings of International Symposium of Multiple-Valued Logic*, pp. 128–135, 1990.

[22] K. Yasuoka, "TACCO: A Programmable Logic Array Optimizer," *Research Report of Kyoto University Data Processing Center*, vol. 6, pp. 45–53, 1991 (in Japanese).

[23] M.A. Heap, W.A. Rogers, and M.R. Mercer, "A Synthesis Algorithm for Two-Level XOR Based Circuits," in *Proceedings of ICCD'92*, pp. 459–462, 1992.

[24] O. Coudert and J.C. Madre, "A New Graph Based Prime Computation Technique," in *Logic Synthesis and Optimization* (edited by T. Sasao), Kluwer Academic Publishers, pp. 33–57, 1993.

[25] T. Sasao, "And-Exor Expressions and Their Optimization," in *Logic Synthesis and Optimization* (edited by T. Sasao), Kluwer Academic Publishers, pp. 287–312, 1993.

[26] K. Yasuoka, "A Generation Method for Exor-Sum-of-Products Expressions Using Shared Binary Decision Diagrams," in *Logic Synthesis and Optimization* (edited by T. Sasao), Kluwer Academic Publishers, pp. 313–322, 1993.

[27] K. McElvain, *IWLS'93 Benchmark Set: Version 4.0*, in *IWLS'93*, 1993.

[28] T. Sasao, "Ternary Decision Diagram and Their Applications," in *IWLS'93*, 1993.

[29] T. Sasao, "EXMIN2: A Simplification Algorithm for Exclusive-OR-Sum-of-Products Expressions for Multiple-Valued Input Two-Valued Output Functions," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 12, 621–632, 1993.

[30] S. Minato, "Zero-Suppressed BDDs for Set Manipulation in Combinatorial Problems," in *Proceedings of ACM/IEEE 30th Design Automation Conference*, pp. 272–277, 1993.

[31] H. Ochi, K. Yasuoka, and S. Yajima, "Breadth-First Manipulation of Very Large Binary Decision Diagrams," in *Proceedings of ICCAD'93*, pp. 48–55, 1993.

[32] Y. Takenaga and S. Yajima, Computational Complexity of Manipulating Binary Decision Diagrams, *IEICE Transactions on Information and Systems*, vol. E77-D, pp. 642–647, 1994.

[33] K. Yasuoka, "A New Method to Represent Sets of Products: Ternary Decision Diagrams," *IEICE Transactions on Fundamentals of Electronics, Communications, and Computer Sciences*, vol. E78-A, pp. 1722–1728, 1995.

Acknowledgements

I would like to express my sincere appreciation to Professor Shuzo Yajima of Kyoto University for his continuous guidance, interesting suggestions, accurate criticisms and encouragements during this research.

I would also like to express my thanks to Associate Professor Naofumi Takagi of Nagoya University who introduced me to the world of logic synthesis, and has been giving me enormous suggestions, criticisms, and encouragements through this research.

I acknowledge interesting comments that I received from Professor Tsutomu Sasao of Kyushu Institute of Technology, Professor Hiroto Yasuura of Kyushu University, Professor Hiromi Hiraishi of Kyoto Sangyo University, Professor Kazuo Iwama of Kyushu University, and Dr. Nagisa Ishiura of Osaka University.

I would like to thank to the members of BDD Workshop for their fruitful discussions, especially to Dr. Shin-ichi Minato of NTT Research Laboratories, Associate Professor Hiroyuki Ochi of Hiroshima City University, Mr. Yasuhito Koumura of Sanyo Corporation, Mr. Noriyuki Takahashi of NTT Research Laboratories, Mr. Masaki Itoh of Hitachi Corporation, Mr. Kazuyoshi Takagi of Nara Institute of Science and Technology, Mr. Hiroshi Sawada of NTT Research Laboratories, Mr. Hiroyuki Higuchi of Fujitsu Research Laboratories, Mr. Seiichiro Tani and Mr. Kazuyoshi Hayase of Tokyo University, and Mr. Koyo Nitta of NTT Research Laboratories.

Thanks are also due to my colleages in Kyoto University Data Processing Center and the members of Professor Yajima's Laboratory of Kyoto University, especially to Associate

Professor Yasuo Okabe, Dr. Kiyoharu Hamaguchi, and Dr. Yasuhiko Takenaga, for their contiguous supports throughout this research.

Lastly, I thank my wife Motoko for her patience, support, and great encouragement.

List of Publications by Author

Book

- K. Yasuoka, "A Generation Method for Exor-Sum-of-Products Expressions Using Shared Binary Decision Diagrams," in *Logic Synthesis and Optimization* (edited by T. Sasao), Kluwer Academic Publishers, pp. 313–322, 1993.

Major Publication

- K. Yasuoka, "A New Method to Represent Sets of Products: Ternary Decision Diagrams," *IEICE Transactions on Fundamentals of Electronics, Communications, and Computer Sciences*, vol. E78-A, pp. 1722–1728, 1995.

International Conference Papers

1. K. Yasuoka, "An Efficient Algorithm for Generating Mixed-Polarity Reed-Muller Expansions Using Shared Binary Decision Diagrams," *International Symposium on Logic Synthesis and Microprocessor Architecture in ISKIT'92*, pp. 196–199, 1992.

2. H. Ochi, K. Yasuoka, and S. Yajima, "Breadth-First Manipulation of Very Large Binary Decision Diagrams," in *Proceedings of ICCAD'93*, pp. 48–55, 1993.

3. K. Yasuoka, "Ternary Decision Diagrams to Represent Ringsum-of-Products Forms,"

in *Proceedings IFIP WG 10.5 Workshop on Application of Reed-Muller Expansions in Circuit Design*, pp. 123–130, 1995.

4. K. Yasuoka, "Ternary Decision Diagrams," in *Proceedings The Fourth International Conference on Computer-Aided Design and Computer Graphics*, Part I, pp. 681–688, 1995.

Technical Reports

1. K. Yasuoka, N. Takagi, and S. Yajima, "An Algorithm for Minimizing Logic Functions Using a Mincube-Adjacency Table," *Technical Report of IEICE on Computation*, COMP88-54, pp. 45–52, 1988 (in Japanese).

2. K. Yasuoka, "TACCO: A Programmable Logic Array Optimizer," *Research Report of Kyoto University Data Processing Center*, vol. 6, pp. 45–53, 1991 (in Japanese).

3. K. Yasuoka, "Optimization of AND-XOR Forms of Logic Functions Using Boolean Derivation," *Research Report of Kyoto University Data Processing Center*, vol. 6, pp. 43–50, 1992 (in Japanese).

4. H. Ochi, K. Yasuoka, and S. Yajima, "Secondary Storage Oriented Breadth-First Manipulation of Very Large Shared Binary Decision Diagrams," *KUIS Technical Report*, KUIS-92-0005, 1992.

5. H. Ochi, K. Yasuoka, and S. Yajima, "A Secondary Memory Oriented BDD Manipulator Using Garbage Collection Based on Sliding Type Compaction," *KUIS Technical Report*, KUIS-92-0007, 1992.

6. H. Ochi, K. Yasuoka, and S. Yajima, "A Secondary Storage Oriented Breadth-First Algorithm for Manipulating Very Large SBDD's," *Technical Report of IEICE on VLSI Design Technologies*, VLD92-73, pp. 25–32, 1993.

Convention Records

1. K. Yasuoka, N. Takagi, and S. Yajima, "Minimization of Logic Functions Using a Mincube-Adjacency Table," in *Record of the 1989 IEICE Spring Conference*, A-238, 1989 (in Japanese).

2. K. Yasuoka, N. Takagi, and S. Yajima, "Exact Minimization of Multiple-Output Logic Function Using Neighfunction," in *Record of 39th IPSJ National Convention*, 7W-3, 1989 (in Japanese).

3. K. Yasuoka, N. Takagi, and S. Yajima, "Logic Function Minimization Using Neighfunctions," in *Record of the 1990 IEICE Spring Conference*, A-104, 1990.

4. K. Yasuoka, N. Takagi, and S. Yajima, "Exact Minimization of Multiple-Output Logic Function Using Neighfunction," in *25th FTC Conference*, 1991 (in Japanese).

5. H. Ochi, K. Yasuoka, and S. Yajima, "A Breadth-First Algorithm for Efficient Manipulation of Shared Binary Decision Diagrams in the Secondary Memory," in *Record of 45th IPSJ National Convention*, 4L-04, 1992 (in Japanese).

6. H. Ochi, K. Yasuoka, and S. Yajima, "A Breadth-First Algorithm for Manipulating Shared Binary Decision Diagrams in Secondary Storage," in *Record of the 1993 IEICE Fall Conference*, A-60, 1993 (in Japanese).

7. K. Yasuoka, "Ternary Decision Diagrams Representing Sets of Products," in *1994FY Winter LA Symposium*, 1995.